This manual describes BASIC language programs for operating the Canon AS-100 System. To realize the AS-100's full potential it is necessary to fully understand Canon BASIC Language.

To make this task as easy as possible, this manual is divided into the following sections.

Chapter I Introduction

This chapter provides beginners with basic information on computers, and how they operate, in addition to defining technical terms. You may skip this chapter if you already have a working knowledge of computers as it contains no essential information about Canon BASIC.

Chapter II Operation

This chapter explains various operations, including system generation and programming of Canon BASIC. It describes all aspects of Canon BASIC for use with the AS-100.

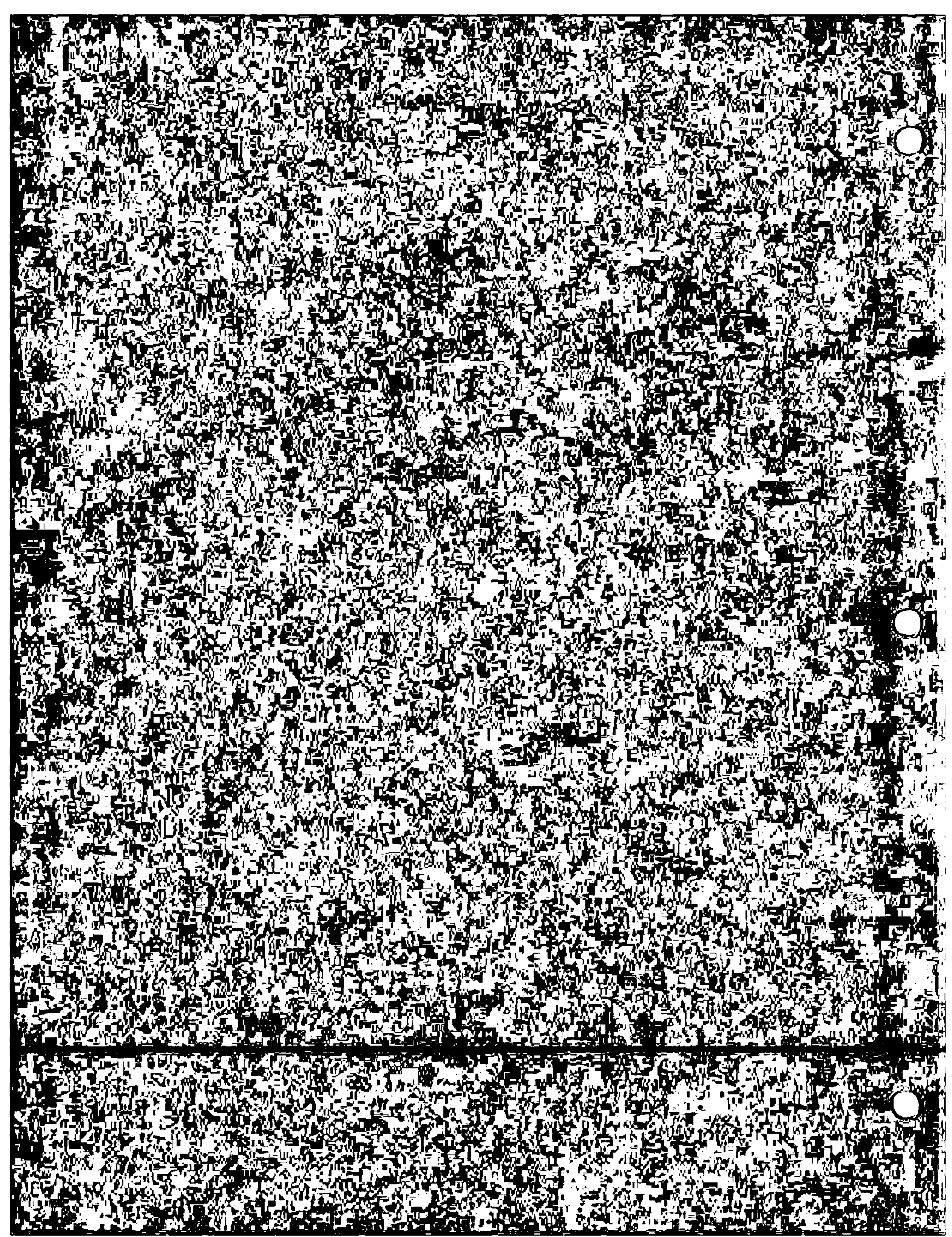
Chapter III Language

This chapter includes Canon BASIC language specifications and details on programming. This section is mandatory for those who will be writing programs in Canon BASIC language.

Chapter I — Introduction

Canon BASIC

Canon AS-100



This chapter provides beginners with basic information on computers, and how they operate, in addition to defining technical terms. You may skip this chapter if you already have a working knowledge of computers as it is contains no essential information about Canon BASIC.

Contents

1. **Basic Construction of the Computer** 1

2. **Data** 4

3. **Programs** 6

4. **Commands** 7

5. **Operating System** 8

6. **Disks** 9

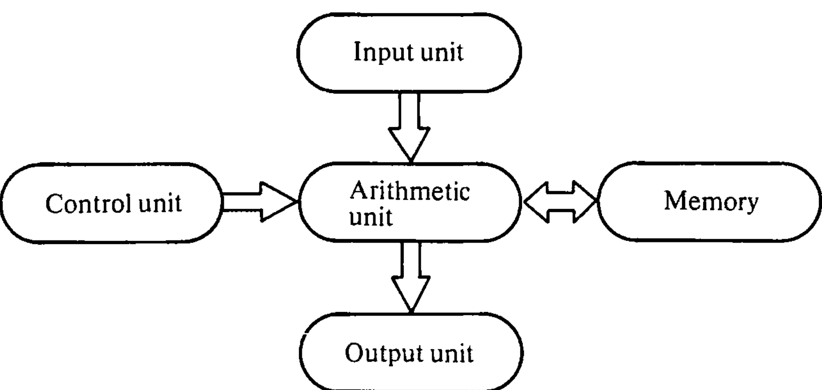
7. **BASIC Language** 10

1. Basic Construction of the Computer

Until about twenty years ago, the word "computer" conjured up images of human-like robots or vast rooms filled with monstrous contraptions that flashed and whirred, things from another place and time. In fact, computers were the very stuff of science fiction.

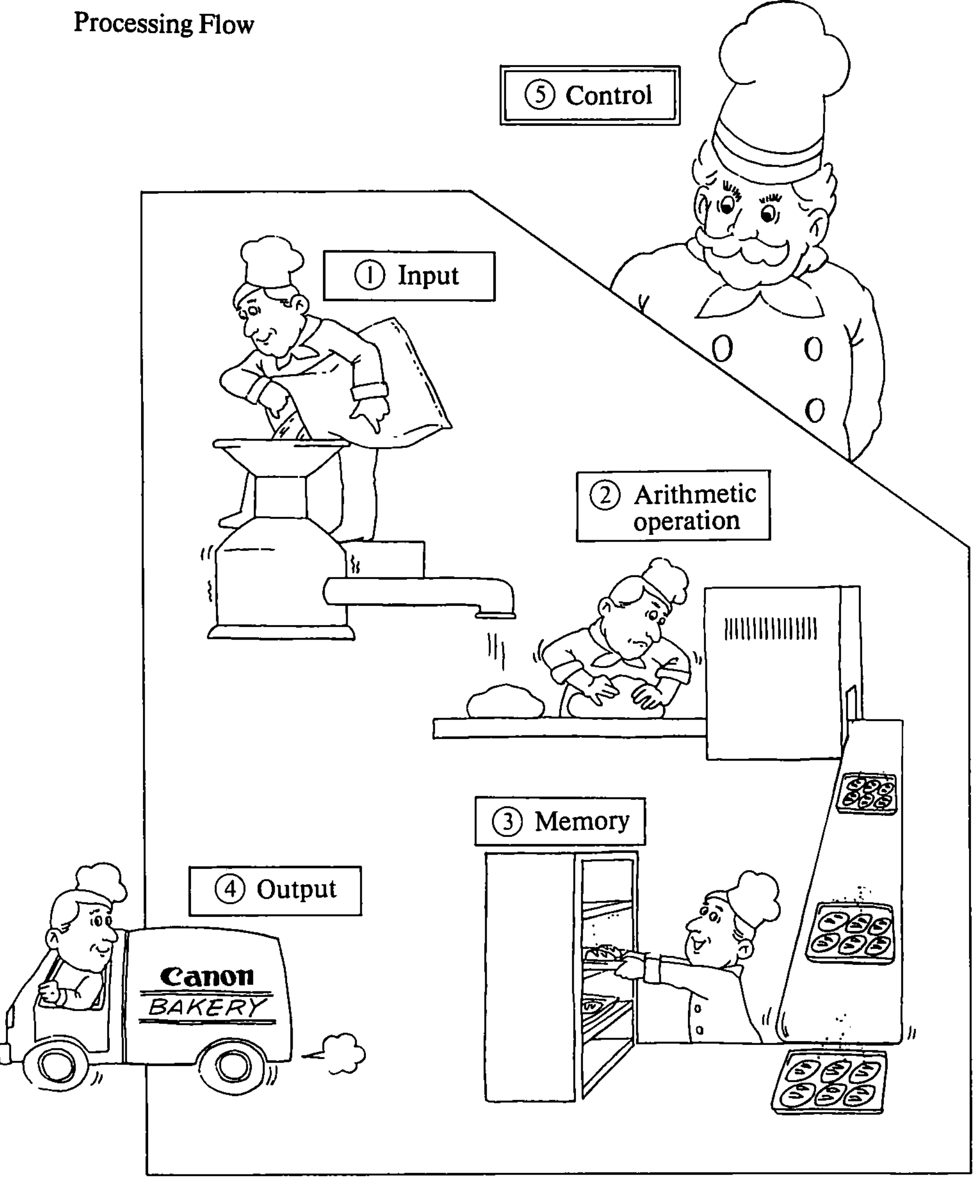
Computers have come a long way since then. Today they touch nearly every aspect of our lives. Microcomputers can be found in automobiles and cameras, television sets and stereos, typewriters and watches. And are being used to automate every field from business and finance to education and medicine. Some are designed to serve a multitude of functions, like the AS-100. All are playing an increasingly vital role in society.

Despite thier varied forms, computers usually have the same basic structure—an input unit, an output unit, a memory unit, an arithmetic unit, and a control unit.



The terms input, output, control, arithmetic, and memory are fundamental to understanding how computers work. To give you a better understanding of these terms, we've illustrated their functions in the following cartoon.

Processing Flow



① The supply of ingredients corresponds to data input. Data that will be processed by the computer is entered or input into the computer.

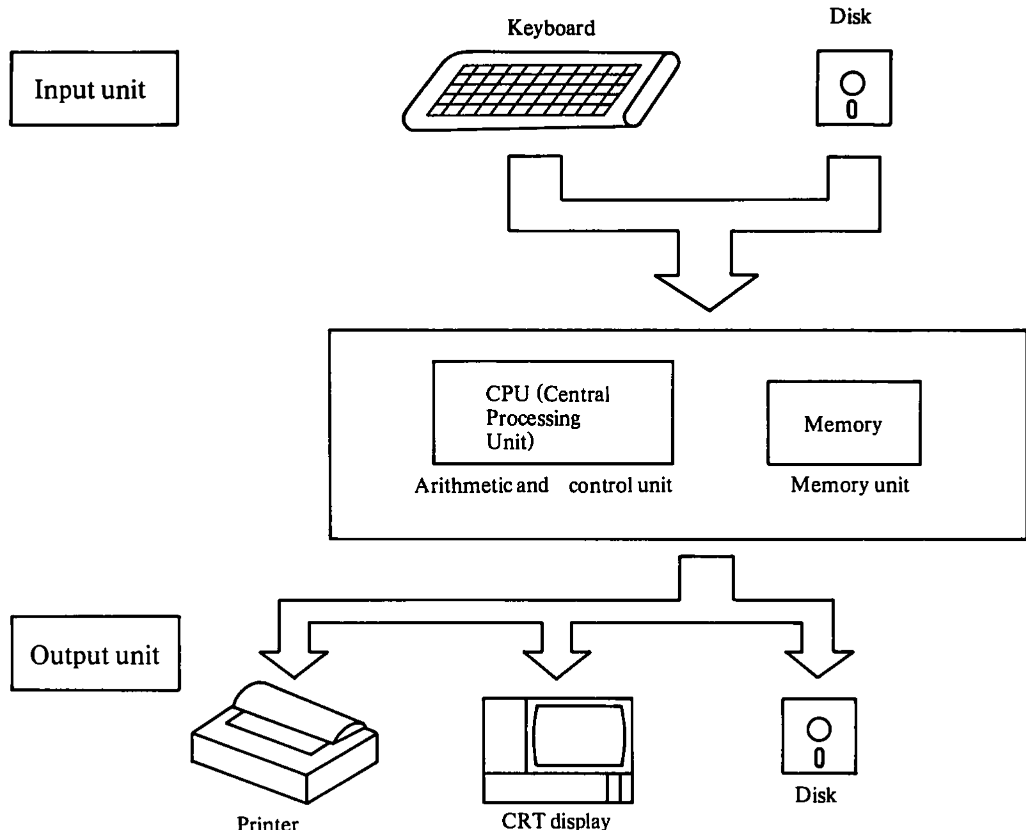
② The place where the ingredients are mixed, kneaded, and finally baked is the arithmetic unit. Actual data processing (calculation, sorting, etc.) is performed inside the computer.

③ The bread is then sent (or stored) in the warehouse, where other ingredients (data) are also stored. This warehouse is the memory or storage unit.

④ Finally the bread is shipped (or output) as a product. This task is handled by the output unit.

⑤ This entire process is supervised by the control unit, from input and arithmetic operation, to storage and output.

Let's take a closer look at the AS-100. Its basic construction is as follows.



- Keyboard .... This is a unit for entering data and operator's instructions.
- Disk .... This is a unit media for recording or storing data magnetically. The disk is classified as an input unit when magnetized data is read from it.
- CPU (Central Processing Unit) ......
  This unit controls all computer functions and performs various processing operations like calculations. The CPU is the core of the computer.

- Memory. . . . . Memory usually means IC memory. This is a temporary storage area for data and programs that will be executed in the computer.
- Printer . . . . This device prints data as readable characters, etc.
- CRT display . . . . . . This unit displays data as readable characters, etc.
- Disk . . . . The disk is classified as an output unit when data is written into it magnetically.

The disk is classfied both as an input unit and an output unit because like a tape recorder, the disk can record data and the recorded data can also be read from the disk. The disk is also called an external storage device.

2. Data

The computer must convert data into electrical signals for processing. Numbers and characters you enter through a keyboard are all converted into electrical pulses called digital signals for transfer into the computer.

Digital signals convey information by their status: either on or off. Within the computer, all processing is performed with the data and signals expressed by two states: "on or off" or "1 or 0."

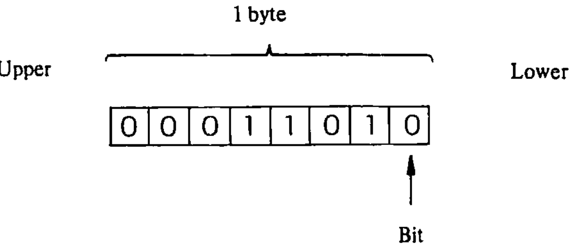
To use a computer, it is hardly necessary for you to understand the signal flow inside the computer. But it is very important to have a clear concept of digital signals to understand the way data is processed in the computer.

The smallest unit of data processed in a computer is a bit. The bit is the unit that can indicates the on or off status. Groups of these bits are used to represent data.

You may wonder how various kinds of data can be represented simply by combining by these states. To understand this, you must first know the concept of binary notation. For example, the number "26" is expressed in decimal notation. This figure is expressed as "11010" in binary notation (in which only 1 and 0 are used). That is, the collection of five bits that can represent 1 or 0 can express the value 26.

Decimal		Binary
26	→	11010

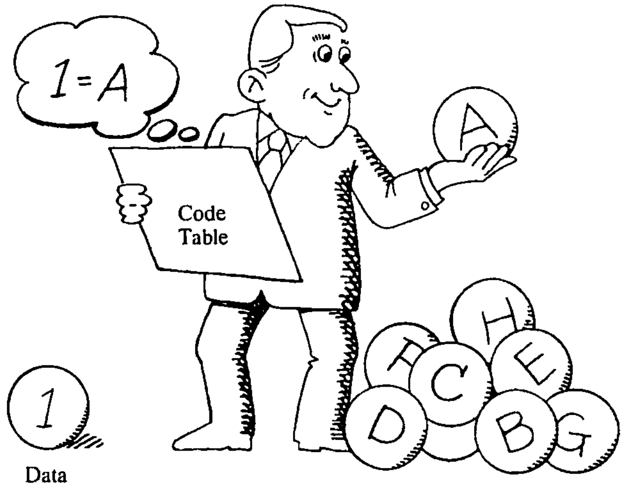
The computer usually processes data in units called bytes. One byte consists of eight bits. One byte can express numeric values from 0 through 255.



Of course, the computer can process characters too.

Characters are expressed by preset codes. For example, if it is decided in advance that the number 1 means "A", the data, 1, expressing the character, is understood as "A". Thus one byte of data can express 256 different characters.

The coding system of the AS-100 is based on standard codes called ASCII codes, which were established within the computer industry.

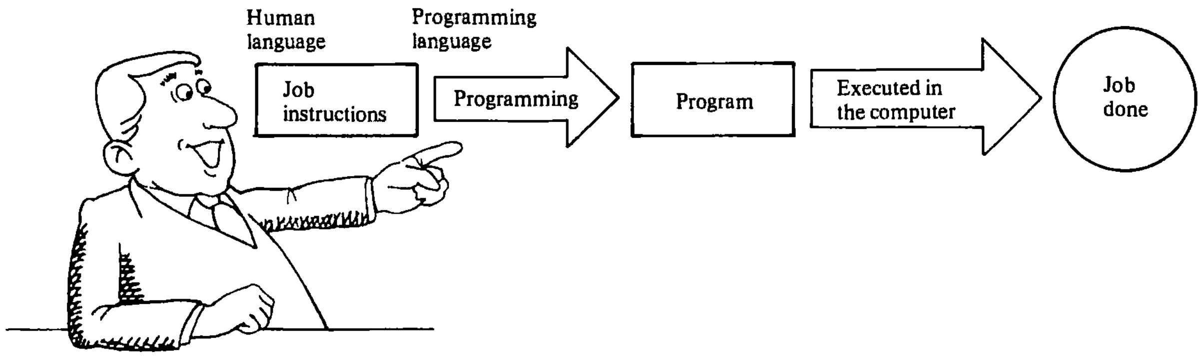


3. Programs

A program is a group of working instructions for the computer. The computer performs actions sequentially as instructed by the program. This is called program execution.

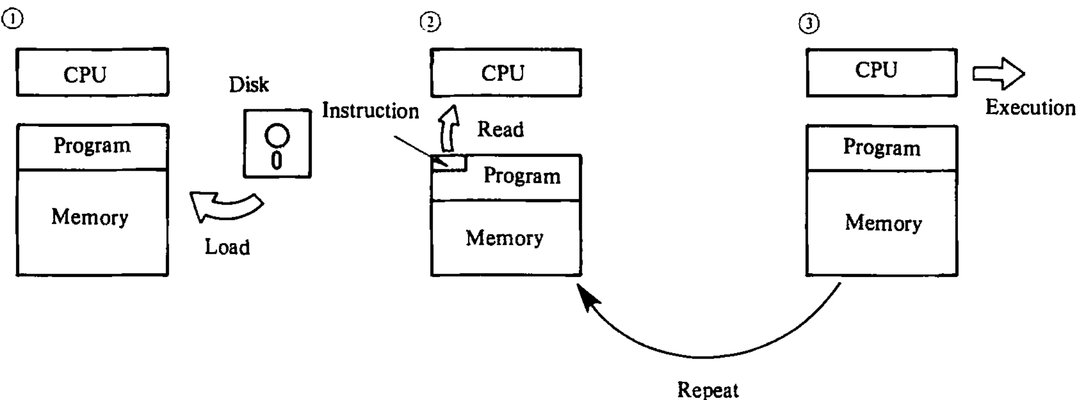
Computers do not understand human language. They only understand instructions given in a particular programming language.

So for the computer to function, you first have to describe the job in a programming language that the computer understands. This step of description is called programming. The computer can only do the job when a program is executed.



How is a program executed in the computer?

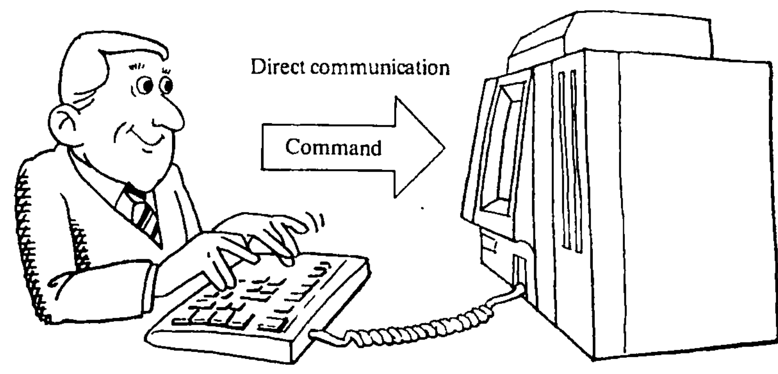
A program can only be executed when it is in memory. For example, when a program is stored on a disk, the program must be first transferred or read to the memory area of the computer. This step is called program loading. After the program is loaded, execution is possible. The computer reads instructions one by one from the beginning (head) of the program in the memory and executes them. Program execution is accomplished by the repetition of this reading and execution sequence.



4. Commands

A program makes the computer perform certain tasks. But how is the program prepared and loaded? There has to be some way to operate the computer while it is not executing a progrm.

The computer can be directed to perform certain routine operations not only by programs, but also by entering commands through the keyboard using a particular format. Basically, commands allow you communicate directly with the computer. The computer also communicates with you through the display.



Here's an example. Let's say you want the computer to execute a certain program.

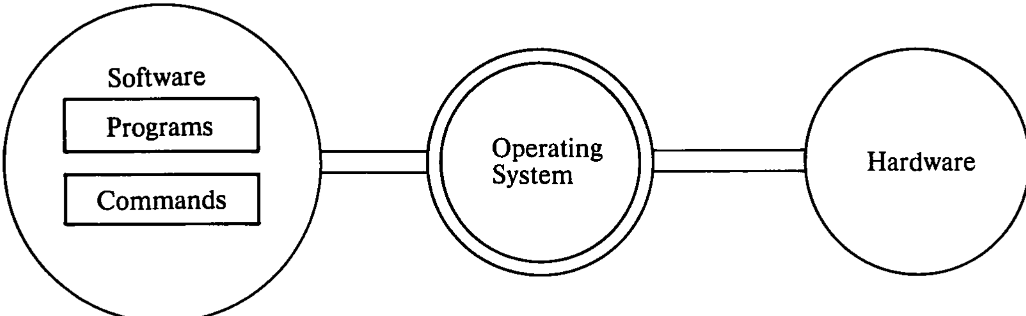
Canon BASIC has a command called "RUN". Look at the display to see if program execution is possible. If it is, depress the R, U, and N keys. You'll see that RUN is displayed on the CRT. This confirms that the command has been entered correctly. But the command is not yet complete. Depress the [ENTER] key, which indicates that command input is complete. After receiving the RUN command, the computer evaluates the command and starts program execution.

5. Operating System

A direction issued by either a program or a command specifies the result of a function like "Display ABC". To realize this function, a procedure consisting of a group of simple action must be performed step by step. The computer first reads the data which indicates the shapes of the characters "ABC" and then sends them to the CRT display circuit. Finally the characters "ABC" are displayed on the screen.

The operating system interprets this kind of direction and indicates the details of the procedure which will carry out the direction according to the hardware specifications. This enables the operator to control the computer with the simplest directions under the support of the operating system.

That is, the operating system is a program that acts as an intermediary between software, like commands and programs, and the individual hardware devices of the computer, thus creating an environment in which the user can use the computer functions easily and effectively. The operating system is sometimes called a system program.



In the AS-100, the operating system is stored on a disk and loaded into memory when the power is turned on. Unless the disk that stores the operating system, called a system disk, is set in the disk drive, the AS-100 cannot be used even if the power is turned on.

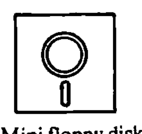
6. Disks

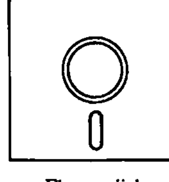
The contents of memory in the computer are deleted when the power is turned off. Therefore, the data and programs have to be stored in disks.

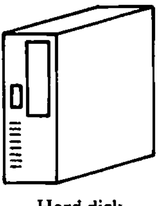
Like recording tapes, disks preserve data magnetically. There are basically 3 types of disks which used with the AS-100—mini floppy disks, floppy disks, and hard disks.

Mini floppy disks and floppy disks are detachable flat circular plates enclosed in square envelopes. When set in devices called disk drives, data can be read from and written to it. Disks 5-1/4 inches in diameter are called mini floppy disks. Those 8 inches in diameter are called floppy disks.

Hard disks are also called fixed disks because data is read from and written to disks fixed on disk drives. There are also various other kinds of magnetic storage devices.


Mini floppy disk


Floppy disk


Hard disk

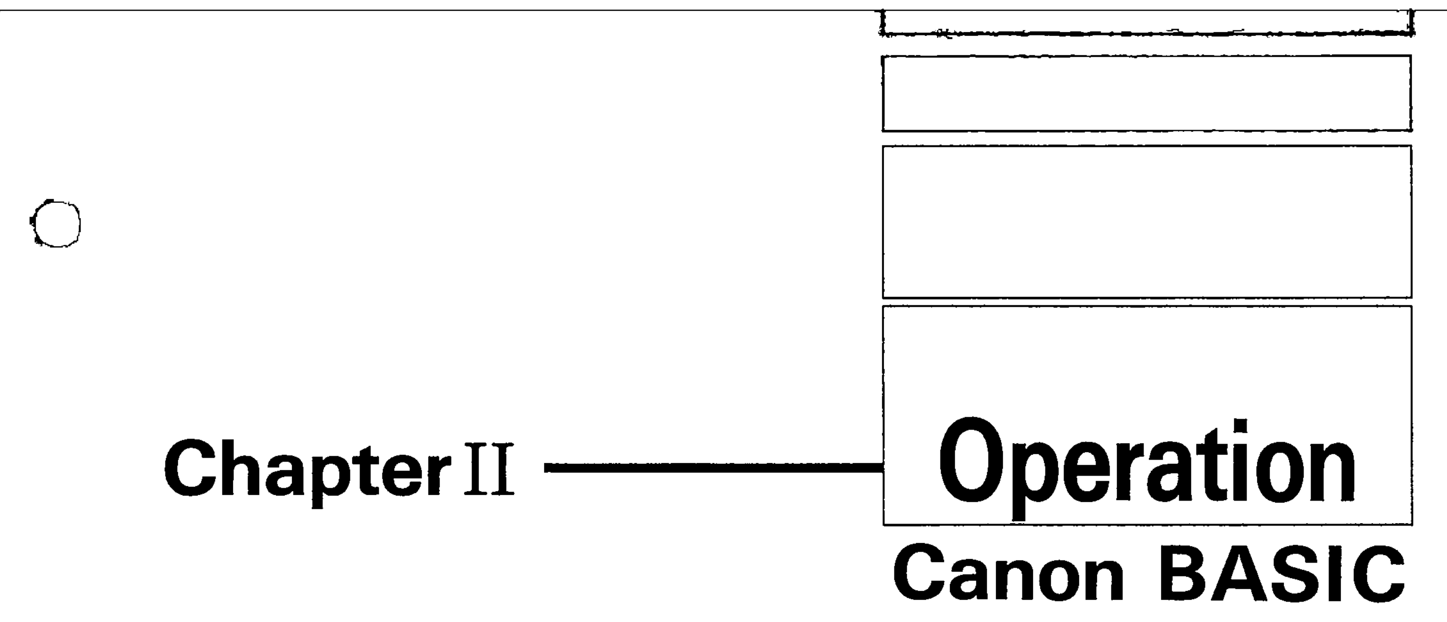
Unlike an audio magnetic tape, a computer disk does not permit data reading and writing unless work called "initialization" is performed. Sometimes called formating, initialization is magnetically dividing the recording surface of the disk into prescribed formats. The computer can only read data from and write it to the disk after initialization.

You can not see a magnetic record on a disk which means a record could be erased by accident. So important programs or data should be stored on more than one disk. Then even if the program or data on one of them is erased, the other disk can be used. Such a reserve disk is called a backup disk. Making backup disks may seem like unnecessary work. But we recommend you always prepare backup disks so that you will not lose important programs or data with a simple mistake.

7. BASIC Language

BASIC is an acronym for "Beginners All-Purpose Symbolic Instruction Code". Since its introduction, BASIC has been used extensively as a high level programming language for microcomputers. BASIC seems to be the easiest programming language for two reasons—because its instructions or commands are easy for beginners to memorize and because it allows versatile processing.

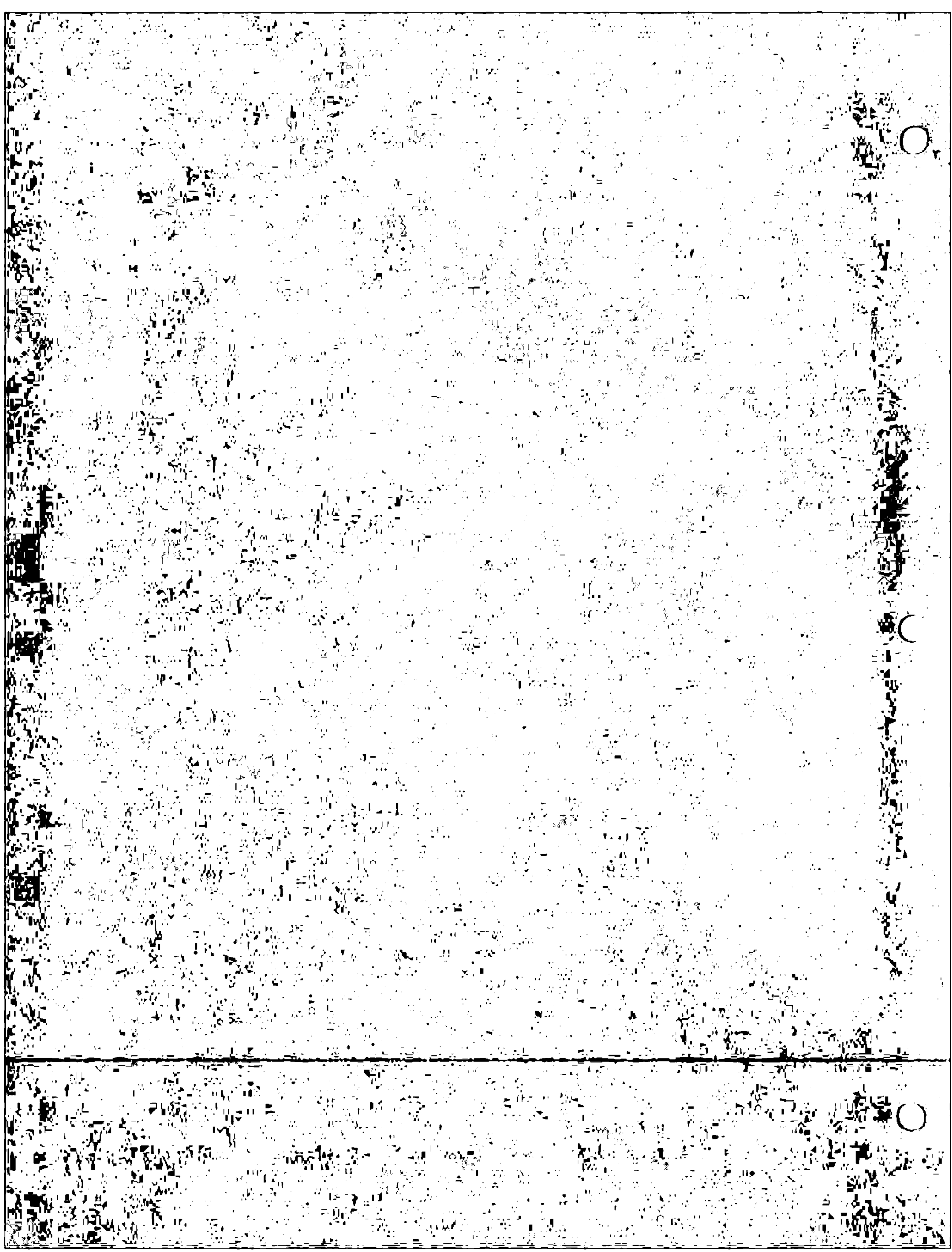
Canon BASIC language, described in Chapter III and elsewhere, is an extended version of BASIC language that has some special instructions and facilities that allow you to use the AS-100's functions easily.



Chapter II — Operation

Canon BASIC

Canon AS-100



This chapter explains various operations, including system generation, commands and programming of Canon BASIC. It describes all aspects of Canon BASIC for use with the AS-100.

Contents

1. An Outline of Canon BASIC 1

1.1 System Outline 1

1.2 Modes 3

1.3 Files 4

1.4 Input/Output Control 5

1.5 Programs 6

1.6 Data 7

2. Use and Operations 8

2.1 System Generation 8

2.1.1 Disk Initialization 10

2.1.2 CP/M-86 Volume Copying 11

2.1.3 Canon BASIC Copying 13

2.2 Hardware Units 14

2.2.1 CRT Display 14

2.2.2 Disk Drives 15

2.2.3 Keyboard 16

2.3 System Start-up 23

2.4 Command Operations 24

2.4.1 Outline and Format of Commands 24

2.4.2 EDIT Command 25

2.4.3 LOAD Command 27

2.4.4 SAVE Command 28

2.4.5 LIST Command 31

2.4.6 XREF Command 33

2.4.7 RUN Command 35

2.4.8 CANCEL Command 37

2.4.9 DLIST Command 38

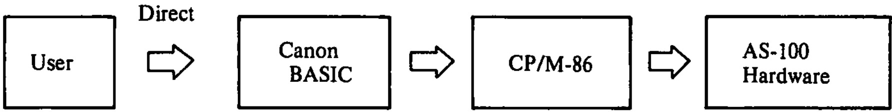
2.4.10 RNAME Command 40
2.4.11 NEW Command 41
2.4.12 BYE Command 41
2.4.13 OS Mode Commands 42
2.4.14 Handlers 51
2.5 Programming 52
2.5.1 Programming Procedure 52
2.5.2 Coding 53
2.5.3 Program Entry 54
2.5.4 Program Editing 58
2.6 Debugging 73
2.6.1 Debugging Mode Outline 73
2.6.2 Debugging Commands 75
2.6.3 Debugging Examples 80
2.7 Functions of Control Key 81

1. An Outline of Canon BASIC

This section gives a general outline of Canon BASIC. For more details, please refer to the explanations in the subsequent sections. In the descriptions that follow, the word BASIC means Canon BASIC.

1.1 System Outline

Canon BASIC operates under the CP/M-86™ operating system. When the AS-100 is operated under Canon BASIC, it uses CP/M-86 indirectly. Conceptually, the functions of Canon BASIC are actually performed by CP/M-86, which BASIC requires to control the hardware.



Canon BASIC facilities can be classified into the following three categories:

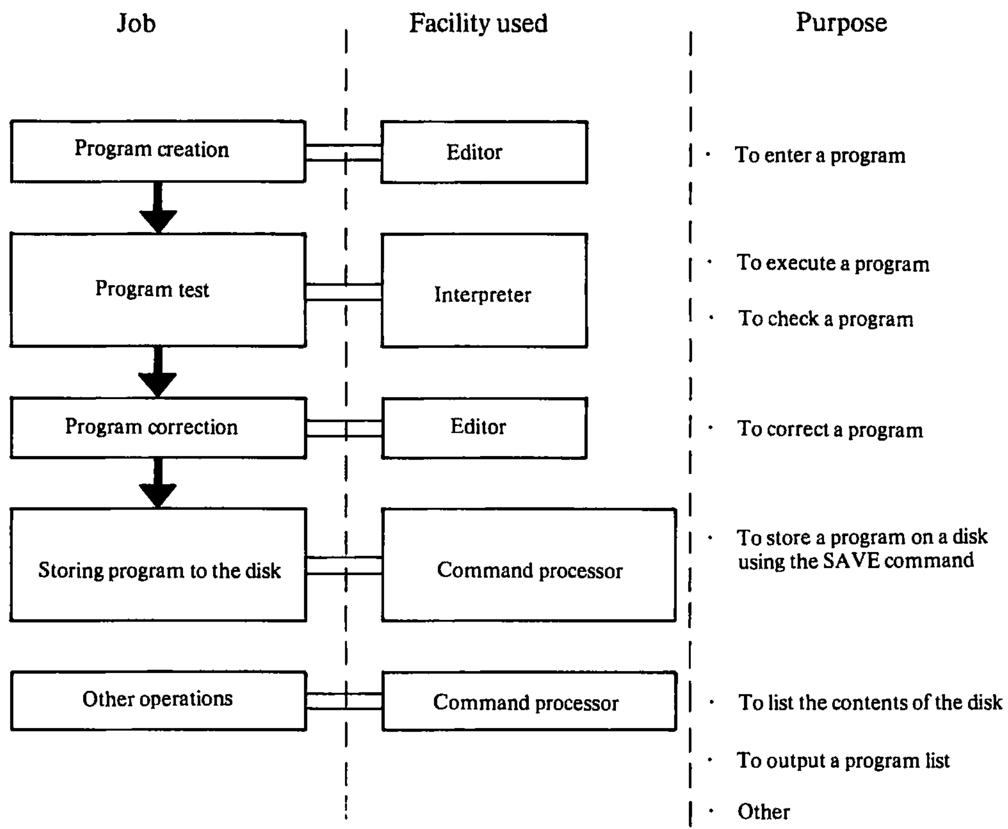
Interpreter. . . . . Interprets a BASIC language program and executes it.

Editor . . . . A program used to create or edit a BASIC program in memory.

Command processor. . . . . .
Interprets commands entered through the keyboard and executes them.

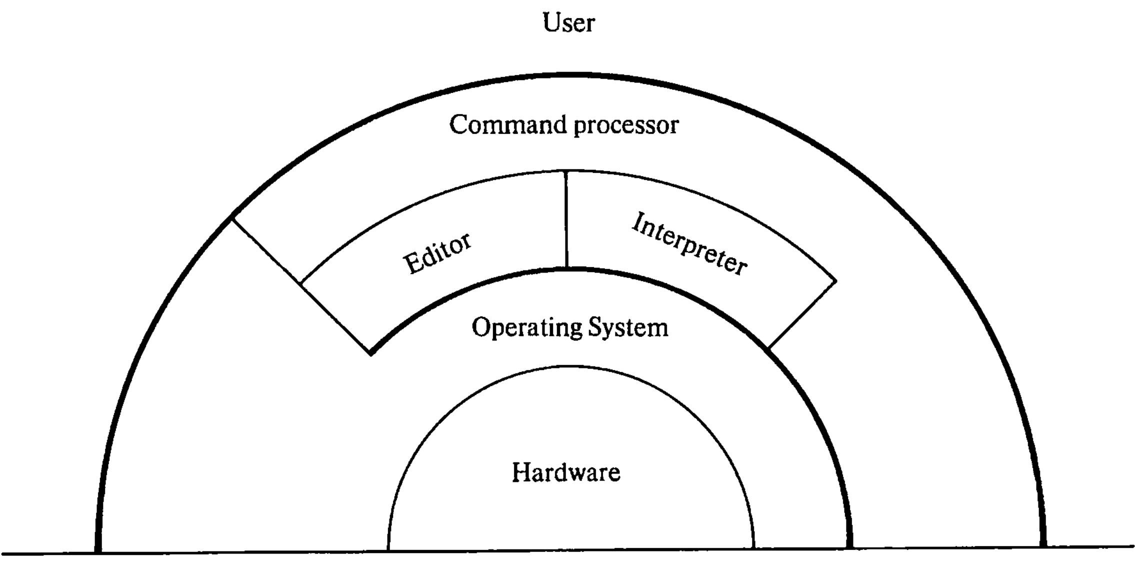
These facilities are actually used as shown in the next page.

* CP/M-86 is a trademark of Digital Research Inc.



The information in the table is merely an example of how the facilities are used. Each will be explained in detail later.

The structure of Canon BASIC is illustrated below.



1.2 Modes

There are two modes, a programming mode and operating mode, in BASIC. The programming mode is for program creation and editing. The operating mode is for execution of commands and programs. Besides, there is an OS mode under the direct control of the OS and a debugging mode which is under control of the operating mode.

Programming mode: When the editor is executed, the AS-100 enters the programming mode. Program creation and editing can be performed in the programming mode. This mode is indicated by prompting "%". When the automatic numbering function of the editor is used, line numbers are displayed instead of "%".

```
10
20
%_
```

Operating mode: Programs and commands can be executed in this mode. The AS-100 is automatically set to this mode when BASIC is started up. This mode is the basic status of Canon BASIC. The operating mode is indicated by prompting "$" on the display.

```
$_
```

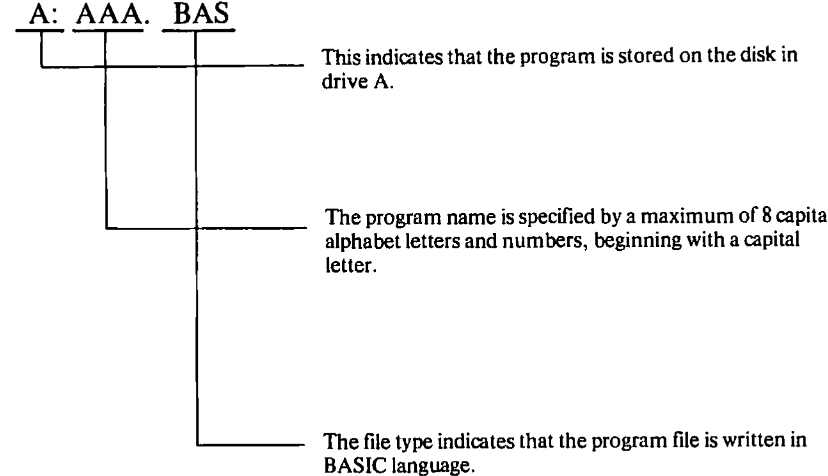
1.3 Files

A file is a program or a block of data recorded on a disk. A file consisting of one program is called a program file. A file consisting of a block of data is called a data file.

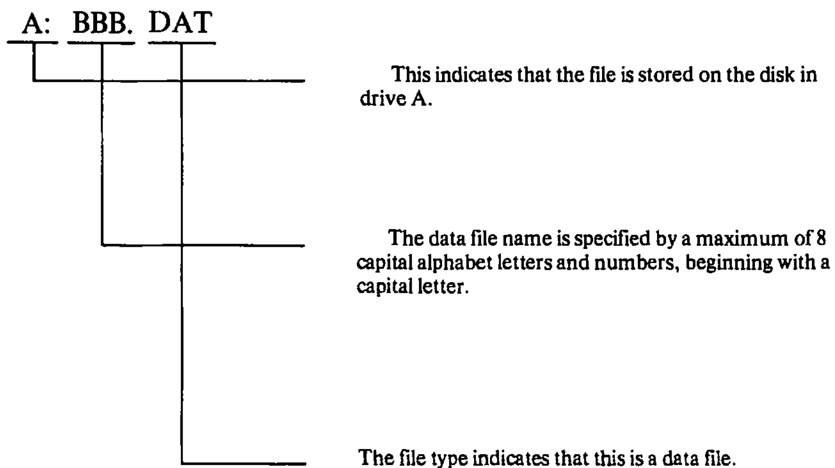
The program is loaded to memory or saved to disk in units called program files. Data read/write from/to the disk is performed against the data file. In program and command operations, files are specified as follows:

<Drive name> : <File name> . <File type>

Program file........ A program file consists of a single program written in BASIC language. The file name is the same as the program name. The file type must be BAS.



Data file...... A data file consists of a block of data which can be used in a BASIC program. During program execution data is read from and written to the data file. The file type must be DAT.



1.4 Input/Output Control

All input and output in the AS-100 under BASIC is controlled by specifying the device name. In BASIC programs, however, device names must be defined as numbers called logical device numbers before execution of input/output instructions. Specify the logical device number in advance with the input/output instructions.

Device names are defined as follows:

Connector No.	RS232C I/F	Centronics I/F
1	×	LPT: or UP0:
2	US0: or TTY:	UL1: or UP1:
3	PTR: or PTP:	UL1: or UP1:
4	US1:	UL1: or UP1:
5	US2:	UL1: or UP1:

The device name CRT: is given to the display and CON: is given to the keyboard. This is specified only when input or output is performed through the display or keyboard using either the GET or PUT statements (explained later).

* The Centronics I/F Board can only be added to connector 3, 4, or 5.

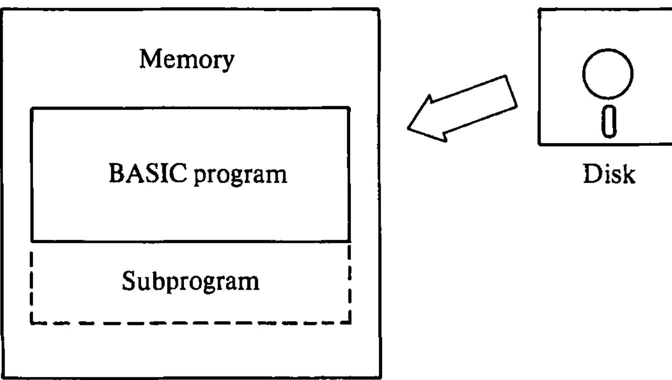
1.5 Programs

The editor is used to create and edit BASIC programs. At this time, a program entered through the keyboard is actually created in memory in an intermediate code which can be interpreted by the interpreter.

It is possible to call and execute a machine language program in a BASIC program.

Only one BASIC program can be loaded (read from the disk to memory) at a time. But two or more BASIC programs can be loaded in memory as subprograms using the CALL statement (explained later).

The size of a BASIC program cannot exceed 32K bytes.



When the program name is omitted in program specification of command operations, the program in memory is automatically specified. This program is called a priority program.

1.6 Data

There are three types of data processed under BASIC. They are real number type, integer type, and string type data. Processing like calculation cannot be performed between different data types. (Processing can be performed between integer type and real number type data.)

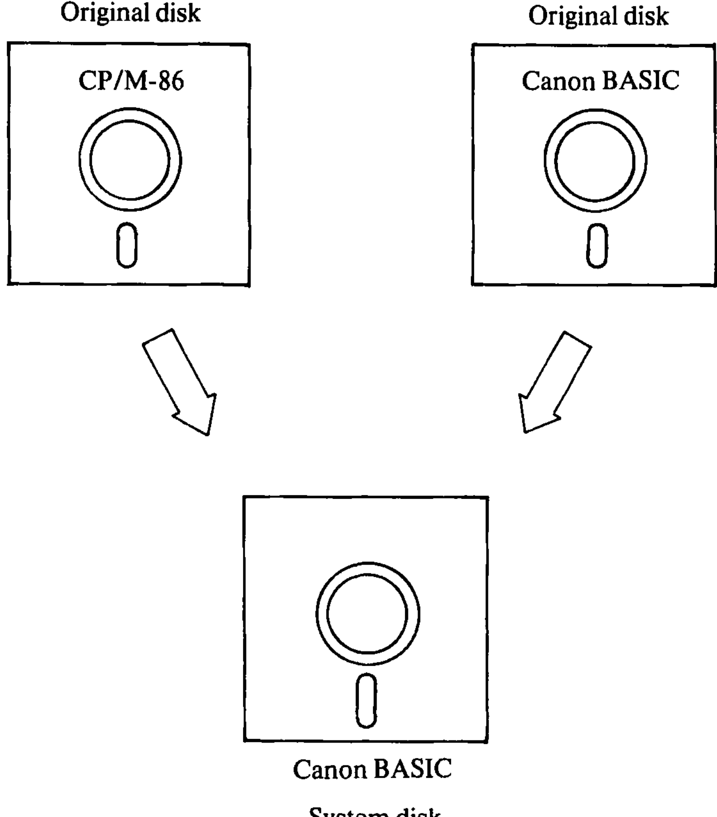
The reading and writing of data from/to a disk is performed using input/output instructions like INPUT, PRINT, GET, and PUT statements. But be careful when using the GET and PUT statements, because the operating system reads/writes data from/to the disk in 128-byte units. Please read the explanations of the statements carefully.

2. Use and Operations

2.1 System Generation

To use BASIC, you first have to generate a system disk for BASIC from the original CP/M-86 disk and the original BASIC disk.

A Canon BASIC system disk, which will be called the system disk in this manual, is generated by copying all of the CP/M-86 system programs onto a blank disk and then copying all of the modules of the Canon BASIC subsystems onto the same disk.



The system generation procedure consists of three steps:

1. Initializing a disk to use as the system disk.
2. Volume copying the CP/M-86 onto the disk.
3. Copying Canon BASIC onto the same disk.

System generation is basically copying the contents of one disk to another disk. This means that a misoperation can erase an important original disk. Observe the following cautions during system generation:

- Read the explanations carefully before performing each operation.
- Do not open the disk drive door except when "A>_" is displayed on the CRT.
- Make sure the disks are set in the correct drives.
- Make sure that you depress the correct keys.

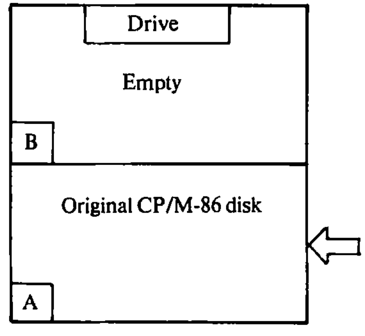
Other remarks:

- Key operations are indicated by a character enclosed by the key mark □. The functions of [↵] and [ENTER] are the same.
- Detailed explanations of the commands used in this operation are given later.
- If an error occurs during operation, refer to the explanation of the corresponding command or "6. Error Messages" in Chapter III.
- After system generation, store the original disks carefully in the correct environment.
- There is no explanation given for system generation involving disks of different shapes, i.e. when an 8-inch Canon BASIC system disk is generated from a 5-inch original disk. If copying to a different-sized disk is necessary, read the explanation of each command and perform the same three steps initializing the disks, copying the CP/M-86, and copying Canon BASIC. The VOLCOPY and the COPYDISK commands cannot be used when copying to disks of a different size.

2.1.1 Disk Initialization

New blank disks must be initialized before they are used with the AS-100. Initialization involves checking a new blank disk and dividing or formating the disk surface into prescribed formats so that data can be written to and read from the disk. The FORMAT command is used to initialize both mini floppy and floppy disks.

- Disk initialization procedure



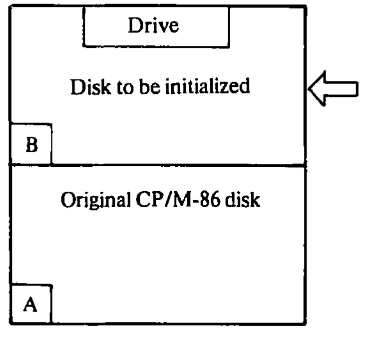
1. Set the original CP/M-86 disk in drive A.

2. Turn the display unit's power on.

3. CP/M-86 is loaded and "A>_" is displayed.

4. Set the disk that will be initialized in drive B.

5. Depress [F][O][R][M][A][T] [↵]



6. The following message is displayed:

```
FORMAT Vx.xx
Disk B:will be destroyed,OK?_
```

7. Depress [Y] and [↵].
   (*If [N] and [↵] are depressed, initialization is not performed and the display returns to "A>_".)

8. `COPYING SECONDARY BOOT` is displayed.

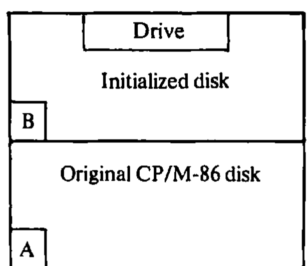
9. When initialization is completed without an error, "A>_" is displayed.

* Initialization for mini floppy disks takes about 40 seconds; floppy disks take about 90 seconds.

2.1.2 CP/M-86 Volume Copying

Volume copying means making a copy of the entire disk. In the following operation, the original CP/M-86 disk is copied.

- Volume copying the original CP/M-86 disk



1. Continued from the disk initialization procedure described 2.1.1.

   Drive A: Original CP/M-86 disk
   Drive B: A new blank disk just initialized.
   Display: A>_

2. Depress [V][O][L][C][O][P][Y][↵]

3. The following message is displayed:

   | Enter Source Disk Drive(A-D)?_ |

4. Depress [A] and [↵] .

5. The following message is displayed:

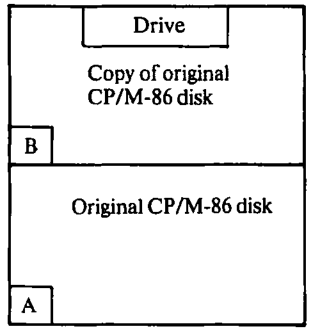
   | Destination Disk Drive(A-D)?_ |

6. Depress [B] and [↵] .

7. The following message is displayed:

   | Copying Disk A:to Disk B:
Is this what you want to do(Y/N)?_ |

8. Depress [Y] and [↵] .
   (*If [N] and [↵] are depressed, volume copying is not performed and the display returns to "A>_")



9. The following message is displayed:

   `COPY TRACK NUMBER=0`

   The track number being checked is displayed.

10. The following message is displayed:

    `Copy another disk(Y/N)?_`

11. Depress [N] and [↵] .

12. When volume copying is completed without an error, "A>_" is displayed.

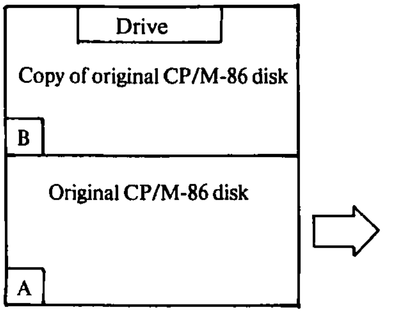
*Volume copying CP/M-86 takes about 2 minutes.

2.1.3 Canon BASIC Copying

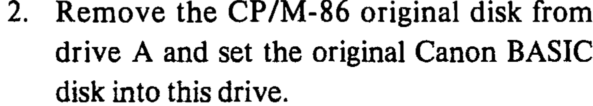
This procedure copies the contents of the original Canon BASIC disk to the disk on which the original CP/M-86 disk is copied.

Unlike volume copying, this operation uses the PIP command of the OS to add the Canon BASIC system program to the CP/M-86 system disk.

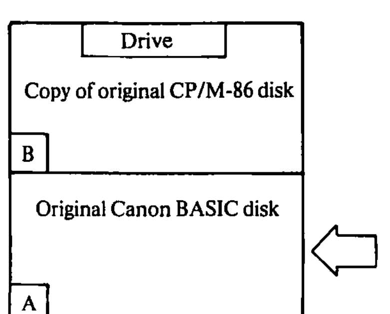
- Copying Canon BASIC onto the CP/M-86 system disk



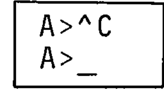
1. Continued from the volume copying operation described in 2.1.2.
   Drive A: Original CP/M-86 disk
   Drive B: Copy of original CP/M-86 disk
   Display: A >_



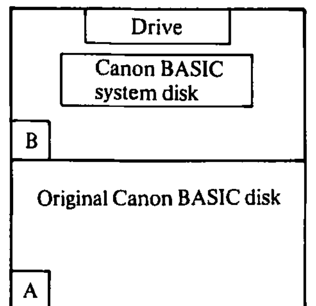
2. Remove the CP/M-86 original disk from drive A and set the original Canon BASIC disk into this drive.



3. Depress [CTRL] and [C] simultaneously.
4. The following is displayed:



5. Depress [B][:][P][I][P] [SPACE] [B][:][=][A][:][*][.][*][↵]



6. When copying is completed without an error, "A >_" is displayed.

   * Copying Canon BASIC takes about 1 minute.

7. System generation is completed. The disk now set in drive B is the Canon BASIC system disk.

2.2 Hardware Units

This section explains the functions of the AS-100 System units under BASIC. For the specifications of hardware devices, please read the "AS-100 System Instruction Manual" included with the display unit and the instruction manual for each peripheral device.

2.2.1 CRT Display

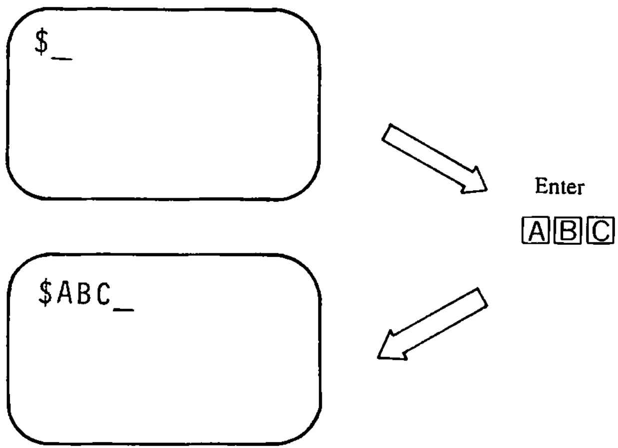
(1) Capacity

The CRT display of the AS-100 has a standard display capacity of 2000 characters (80 characters × 25 lines)—alphabetic letters, numbers, signs, and symbols. This is equivalent to 640 × 400 dots.

(2) Cursor

Each input through the keyboard is displayed on the screen. This is called input echo back. The cursor indicates the position where the input echo back and data output are performed on the screen. Input can be made through the keyboard when the cursor is displayed on the screen.

The cursor in the AS-100 is "_". Each input is made at the cursor position.



(3) **Promptings**

A prompting is a symbol displayed just in front of the cursor to indicate the current system mode. The following promptings are displayed for the different modes.

$ .... BASIC operating mode

% .... BASIC programming mode

* When the automatic numbering function of the editor is used, line numbers are displayed instead of the promting.

A> .... OS mode under the direct control of the operating system

@ .... Debugging mode (See 2.6 Debugging.)

2.2.2 Disk Drives

(1) **Drive Names**

The drives of the mini floppy and floppy disk units are defined by drive name. Disks are specified by the drive name in the AS-100 System.

In the mini floppy disk unit, the lower drive is defined as A and the upper drive as B. In the floppy disk unit, the left drive is defined as A and the right drive as B.

When there are two floppy disk units or when there is one mini floppy disk unit and one floppy disk unit, the drive names of one of the two units must be changed to C and D. Consult your Canon sales representative about changing the drive names.

(2) **Current Drive**

If disk specification is omitted in a command operation or a program, drive A is automatically specified. In this case, drive A is called current drive.

(3) **Disk Replacement**

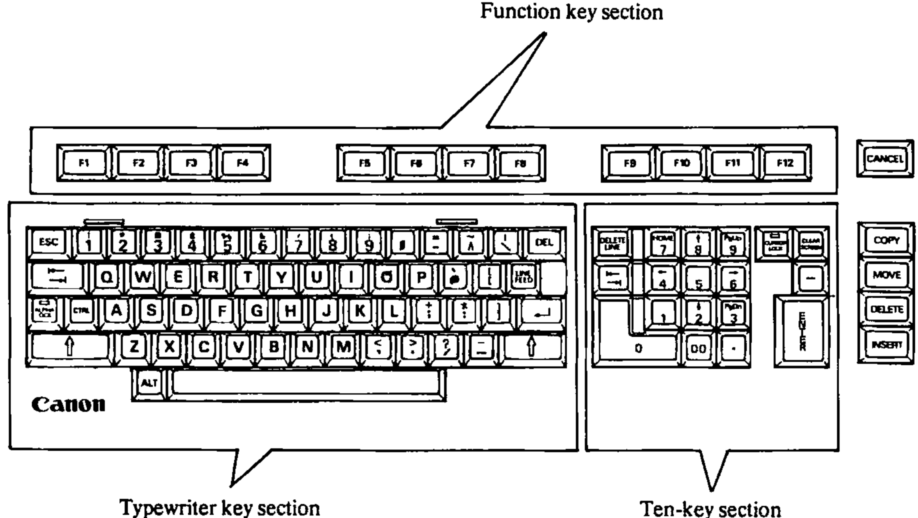
The removal or setting of disks under using BASIC must always be carried out in the command waiting state (see "2.4 Command Operations"). The prompting "$_" is displayed to indicate the command waiting state.

During program execution, disks can be replaced when a message is displayed by execution of the CHANGE statement (described later).

Disk replacement at any other time, may cause destruction of the disk.

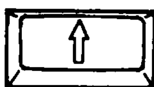
2.2.3 Keyboard

The AS-100 keyboard is divided into three sections by function as shown below.



(1) Typewriter Key Section

The character inscribed on each key is entered depending on the conditions set by the following keys that select the input modes:

	Alpha lock key:	Sets the keyboard in the alpha lock mode. Key depression alternately sets and releases the alpha lock mode. The key lamp lights in the alpha lock mode and goes out when the mode is released.
	Shift key:	Sets the shift mode.

- Normal mode: The alpha lock key lamp is off and the shift key is not depressed. Lowercase (small) alphabet letters, numbers and symbols can be entered.

Ex. [!/1] ⟹ 1 [A] ⟹ a

- Alpha lock mode: Uppercase alphabet (capital) letters, numbers, and symbols can be entered.

Ex. [!/1] ⟹ 1 [A] ⟹ A

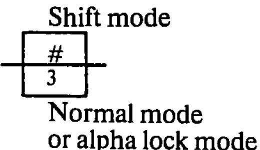
- Shift mode: Uppercase alphabet (capital) letters and symbols can be entered.

Ex. [!/1] ⟹ ! [A] ⟹ A [+/;] ⟹ +

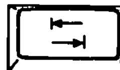
Summary 1

Normal mode → a
Shift mode → A
Alpha lock mode → A

Summary 2



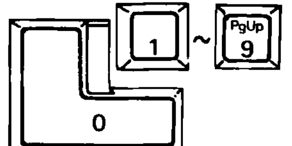
Other keys

	DEL key:	Deletes the character immediately preceding the cursor and moves the cursor one column to the left.
	Tab key:	Moves the cursor to the end of the current input line. In the shift mode, it moves the cursor to the beginning of the current input line.
	Control key:	Used when a console control code (described later) is entered.
	CR key:	This is depressed at the end of a line. Each command is executed when this key is depressed. It has exactly the same function as the ENTER key.
	Space bar:	Enters a space at the cursor position.

(2) **Ten-key Section**

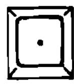
Keys in the ten-key section are used to input numbers and move the cursor.

Delete line key: Deletes the current input line and returns the cursor to the beginning of the line.



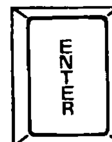
Numeric keys: Enter numbers 0 through 9. In the cursor control mode, these keys control the cursor.

Double zero key: Enters two zeros at once. Depressing this key is the same as depressing the [0] key twice.



Decimal point key: Enters a decimal point.

Minus key: Enters a minus sign.



Enter key: This is depressed at the end of a line. Each command is executed when this key is depressed. It has exactly the same function as the CR key.

Cursor lock key: Sets the numeric keys in the cursor control mode. Each key depression alternately sets and releases the cursor control mode. The key lamp lights in the cursor control mode and goes out when the mode is released. In the cursor control mode, the cursor moves one column at a time in the direction of the arrow printed on the top of the [2], [4], [6] and [8] keys.

(3) Function Key Section

This section consists of 12 function keys. A character string entered with these keys when [⇧] is not depressed (shift down state) can be defined into a required character string in BASIC programs.

Initially the 12 function keys are defined as follows:

F1	:	This key allows one-key, one-instruction input (described later) when used in combination with the typewriter keys.
F2	:	EDIT ␣
F3	:	LOAD␣
F4	:	SAVE ␣
F5	:	LIST ␣
F6	:	XREF␣
F7	:	RUN ␣
F8	:	CANCEL ␣
F9	:	DLIST ␣
F10	:	RNAME ␣
F11	:	NEW
F12	:	BYE

* The symbol " ␣ " in character strings represents a space.

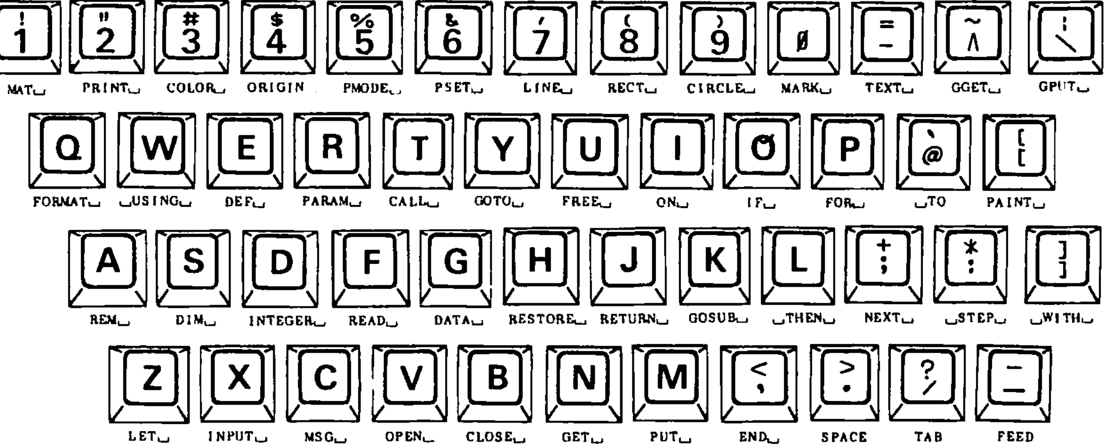
(4) **Other Keys**

CANCEL	Cancel key:	Stops BASIC program execution and sets the system in the command waiting state.
DELETE	Delete key:	This key deletes the character at the cursor position and shifts the following character string one character to the left.
INSERT	Insert key:	Inserts a character string, entered after this key is depressed, at the cursor position. The insertion is completed with the cursor movement operation or by depressing any key but a character key.

(5) **One-Key, One-Instruction Function**

The one-key, one-instruction function can be used to input the keywords during program entry or editing.

You can enter the keywords shown below by depressing the corresponding typewriter key after depressing ~~the~~ [F 1] key.



(6) **Repeat Function and Click Tone**

All typewriter keys and cursor control keys ([↑], [↓], [→] and [←]) are equipped with a repeat function. This means that the input or cursor movement is repeated as long as a key is depressed.

Each key on the keyboard produces a click tone when depressed to confirm entry.

(7) **Key Buffer**

Keyboard input is transfered to the system through the key buffer. The key buffer can store codes of up to 128 characters. Although you do not need to consider the key buffer for ordinary inputs, there are some instructions related to the contents of key buffer.

(8) **Key Operations in this Manual**

Key operations are abbreviated in this manual as follows:

1. The character or symbol entered by a key operation is indicated by enclosing it by □.

   Ex. In the cursor control mode: [→ 6] ⟹ [→]

   In the normal mode: [A] ⟹ [a]

2. [↵] and [ENTER] are indicated as [↵].

3. Sequential key operations are joined by "-".

   Ex. [F1] and [! 1] are depressed one after another. ⟹

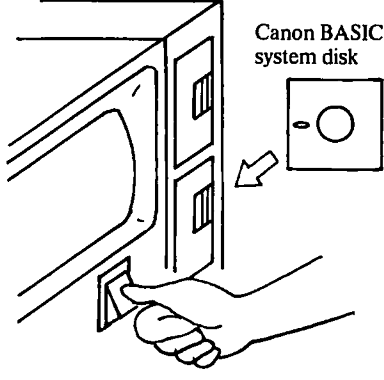
   [F1] - [1]

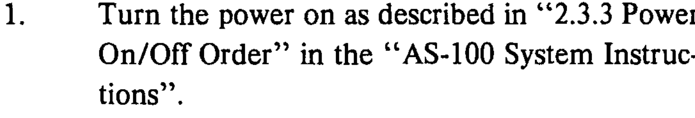
4. Two keys depressed simultaneously are written with a "/" between them.

   Ex. [CTRL] and [S] are depressed simultaneously. ⟹ [CTRL] / [S]

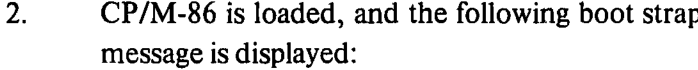
2.3 System Start-up

To start-up the Canon BASIC System, follow the procedure below.

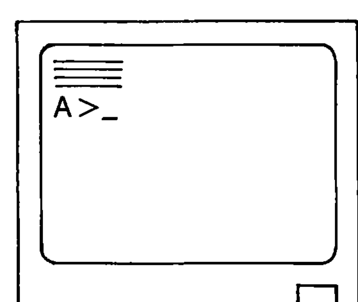


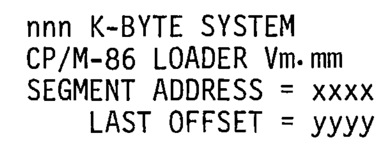


1. Turn the power on as described in "2.3.3 Power On/Off Order" in the "AS-100 System Instructions".
   System disk as used in the manual means the Canon BASIC system disk.



2. CP/M-86 is loaded, and the following boot strap message is displayed:

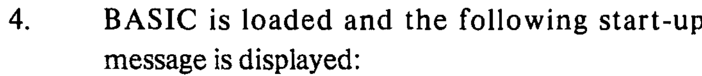




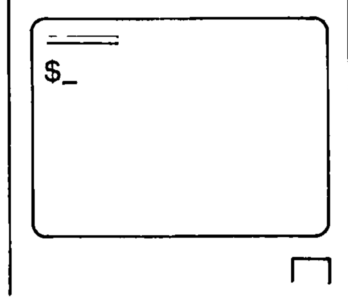
```
nnn K-BYTE SYSTEM
CP/M-86 LOADER Vm.mm
SEGMENT ADDRESS = xxxx
    LAST OFFSET = yyyy

Canon AS-100 CP/M-86 Version z.zz
Copyright(C)1981,Digital Research Inc.
BIOS(A) Vp.pp by Canon Inc.
A>_
```

* Refer to the "CP/M-86 User's Manual" for the CP/M-86 boot strap message.

3. Depress [B][A][S][I][C][↵] .



4. BASIC is loaded and the following start-up message is displayed:




```
Canon AS-100 BASIC Vn.nn
Copyright by Canon Inc.
User's Memory mmmmmmm  Bytes
$_
```

5. BASIC System start-up is completed and commands can now be entered.

Refer to "2.4.13 OS Mode Commands" for the details of the preceding operation.

2.4 Command Operations

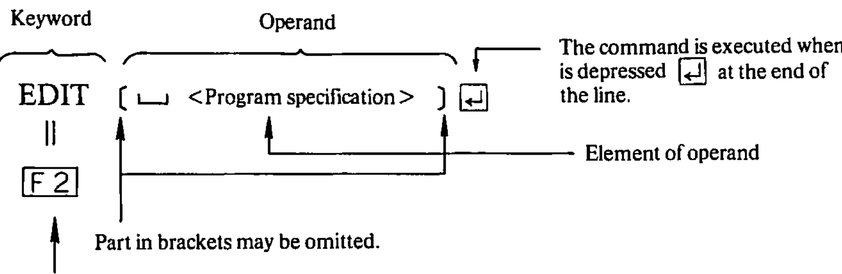
The BASIC system is operated by entering commands through the keyboard. Commands indicate the directives of specific actions and order the AS-100 perform the actions.

Like program instructions, each command has a specific format. Commands can only be entered in the operating mode when prompting "$" and the cursor are displayed. This is called the command waiting state.

2.4.1 Outline and Format of Commands

There are 11 commands under BASIC. Each command consists of a keyword and an operand. By entering the command through the keyboard, the specified action is performed. The format is interpreted as shown in the next example.

Example:



(1) Operands

<Program specification> .... The program file is specified by the drive name, program name, and file type.
Ex. A: ABC.BAS

<File specification> .... The file name is specified by the drive name, file name, and file type.
Ex. A: XYZ.DAT

<Drive specification> .... The drive name is specified by A:~D:.

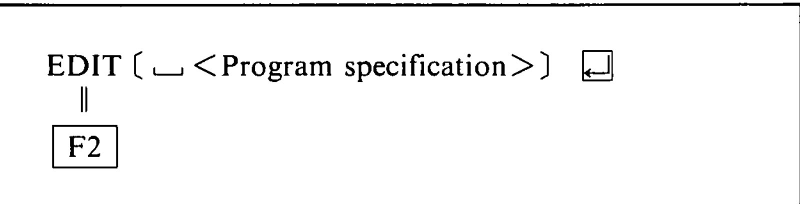
(2) **Notes to Command Operations**

- No distinction is made between uppercase and lowercase letters in command operations.
- The default value for specifications of the drive name, program name, file name, and file type differ from one command to anther.

2.4.2 EDIT Command

The EDIT command activates the editor for program creation or editing. This command cannot be executed for a secured program (explained later).

Format



<Program specification> .... The name of program that will be created or edited is specified. If the program name is omitted, the priority program in memory is automatically specified. If the drive name is omitted, the current drive is automatically specified. The file type BAS may be omitted.

- When the specified program is not in memory or on the disk (i.e. when the program will be created), the following message is displayed:

```
PROGRAM CREATION xxxxxxxx
```

* xxxxxxxx: Program name

The system then enters the programming mode. At this time, "10 _" is displayed by the automatic numbering function of the editor.

- When the specified program is on the disk (i.e. when the program will be edited), the following message is displayed:

```
PROGRAM EDITION xxxxxxxx
```

* xxxxxxxx: Program name

The system then enters the programming mode. At this time, the first 10 lines of the program are listed and "%_" is displayed.

Note: When the EDIT command is executed, the specified program is searched on the disk first even if it is in memory. So when the program specified in the EDIT command is in memory, the program in memory is cleared when the command is executed.

For the details of program creation and editing, refer to "2.5 Programming."

Example 1

«Display»	«Key Operation and Explanation»
	•Create program "ABC".
$_	
	[E][D][I][T] [SPACE] [A][B][C][↵]
PROGRAM CREATION ABC	
10_	

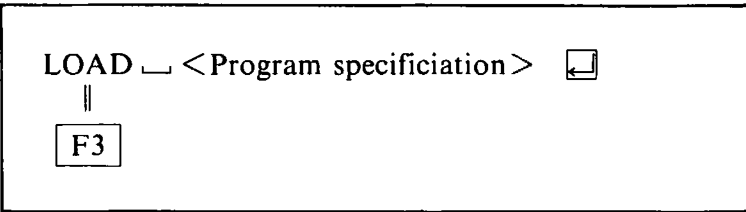
Example 2

«Display»	«Key Operation and Explanation»
	•Edit program "XYZ" stored on the disk in drive B.
$_	
	[E][D][I][T] [SPACE] [B][:][X][Y][Z][↵]
PROGRAM EDITION XYZ	
10	The first 10 lines of the program are listed.
~	
100	
%_	

2.4.3 LOAD Command

The LOAD command loads a specified program from the disk to memory. A program must be loaded in memory when the XREF command (described later) and the LIST command are used, so the program is loaded from the disk to memory in advance using this command.

Format



<Program specification> .... The program that will be loaded to memory is specified. When the drive name is omitted, the current drive is automatically specified. The file type BAS may be omitted. The program name must be specified. If the program specified is not on the disk, the message "<Program specification> NOT EXIST!" is displayed. Loading is not executed and the system returns to the command waiting state.

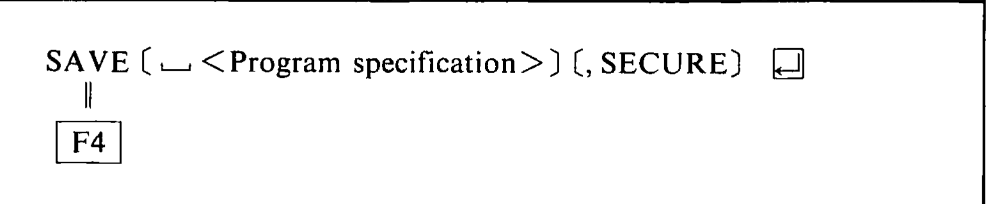
Example 1

«Display»	«Key Operation and Explanation»
	•Load program "ABC" stored on the disk in drive B to memory.
$_	
	[L][O][A][D] [SPACE] [B][:][A][B][C][↵]
$_	

2.4.4 SAVE Command

The SAVE command stores the program in memory to the disk. The program in memory is not deleted. When a program is stored or saved to the disk, the program can be "secured" so that it cannot be specified in the XREF, LIST, and EDIT commands. If there is no program in memory, executing this command causes an error.

Format



<Program specification> .... The name of the program stored on the disk is specified. The name specified does not have to be the same as that of the program currently in memory.
When the program name is omitted, the name of the program in memory is automatically specified. When the drive name is omitted, the current drive is automatically specified. The file type BAS may be omitted.

, SECURE .... When this is specified, the XREF, LIST, and EDIT commands cannot be executed for the program stored in the disk. (The program in memory is not secured.) If the XREF, LIST, or EDIT command is issued to a secured program, the message: "SECURED PROGRAM!" is displayed and the command is not executed. Once a program is secured, it cannot be released.

When this command is entered, the following confirmation message is displayed:

SAVE␣TO␣<Program specification> ␣ (Y/N)?_

When [Y] and [↵] are depressed, the program is stored. When [N] and [↵] are depressed, the command is not executed and the system returns to the command waiting state. When a program of the same name as the program specified is already stored on the specified disk, the following message is displayed:

CANCEL␣OLD <Program specification>␣(Y/N)?_

When [Y] and [↵] are depressed, the program already on the disk is deleted and the specified program is stored. When [N] and [↵] are depressed, the command is not executed and the system returns to the command waiting state.

Example 1

«Display»	«Key Operation and Explanation»
	•Store program "ABC" in memory to the disk in drive A using the same program name.
$_	
	[S][A][V][E][↵]
SAVE TO A:ABC.BAS (Y/N)?_	
	[Y][↵]
$_	

Example 2

«Display»	«Key Operation and Explanation»
	•Store the program in the memory to the disk in drive B using program name "XYZ".
$_	
	[S][A][V][E] [SPACE] [A][:][X][Y][Z][↵]
SAVE TO B:XYZ.BAS (Y/N)?_	
	[Y][↵]
$_	

Example 3

«Display» «Key Operation and Explanation»

- Delete program "EFG" on the disk in drive A and store the program in memory to the disk in drive A using the same program name.

```
$_
                    [S][A][V][E] [SPACE] [A][:][E][F][G][↵]
SAVE TO A:EFG.BAS (Y/N)?_
                    [Y][↵]
CANCEL OLD A:EFG.BAS (Y/N)?_
                    [Y][↵]
$_
```

Example 4

«Display» «Key Operation and Explanation»

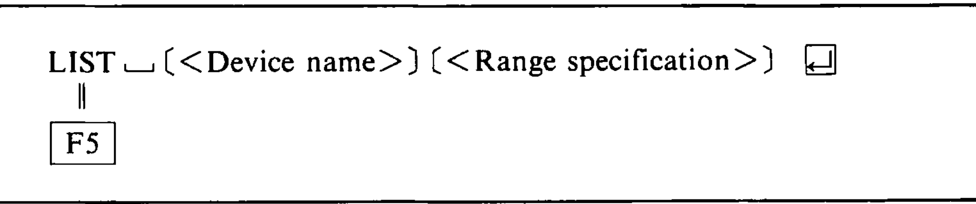
- Secure and store program "POR" in memory to the disk in drive A.

```
$_
                    [S][A][V][E] [SPACE][,][S][E][C][U][R][E][↵]
SAVE TO A:POR.BAS (Y/N)?_
                    [Y][↵]
$_
```

2.4.5 LIST Command

The LIST command outputs the list of a program in memory to the specified device. This command can also specify the the range of program lines that will be listed. This command cannot be used for secured programs. If there is no program in the memory, executing this command causes an error.

Format



<Device name> .... The name of the device to which the list will be output is specified. For device names, refer to "1.4 Input/Output Control." When the device name is omitted, the display (CRT) is automatically specified. When one of the disk drives (A:~D:) is specified as the output device, a list file is created on the disk in the specified drive. (Refer to "2.5 Programming.") The name of the list file consists of program name plus the file type LST.

<Range specification> .... The range of the list that will be output is specified by line number. It is specified as follows:

No specification... All program lines are listed.
<Line number>... Only the specified line is listed.
<Line number>,... All lines from the specified line through the last line of the program are listed.
,<Line number>... All lines from the beginning of the program through the specified line are listed.
<Line number>,<Line number>...
All lines from the first line specified through the last line specified are listed.

The following header is always output at the head of a list:

```
ABC.        1372
 ↑            ↑
Program name  Program size (bytes)
```

If the line specified to indicate the beginning of the output range is not found in the program, listing starts from the line immediately following the line specified. If the line specified to indicate the end of the output range is not found in the program, listing ends with the line immediately preceding the line specified. If only one line is specified and it is not found in the program, only the header is output. The output of a list can be aborted by depressing [CANCEL] or [CTRL] / [C] .

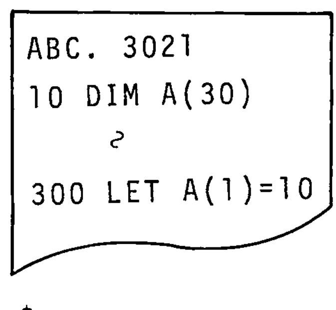
Example 1

«Display»	«Key Operation and Explanation»
	•Output Lines 10~30, of the program in memory to the display.
$_	[L][I][S][T] [SPACE] [1][0][,][3][0][↵]

```
ABC. 3021 ←Program name and size
10 DIM A(30)
     ~           Program contents from Lines 10~30
30 LET A(1)=10
```

Example 2

«Display»	«Key Operation and Explanation»
$_	•Output Lines 10~300, of the program in memory to the printer connected to connector 1.
Printout	[L][I][S][T] [SPACE] [L][P][T][:][1][0][,][3][0][0][↵]

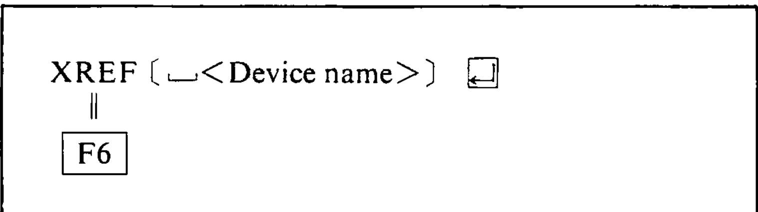


$_

2.4.6 XREF Command (Cross Reference)

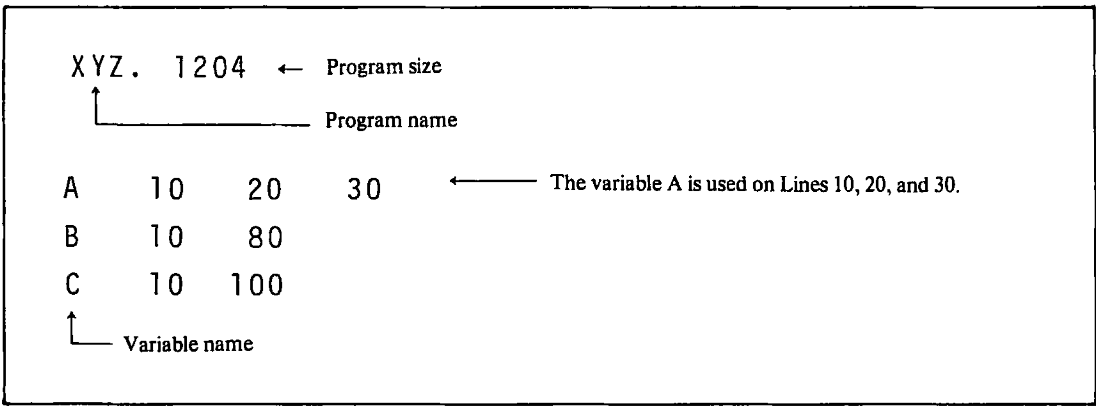
The XREF command outputs a list of the variable names used in the program in memory. The list shows the line numbers where the variables are used. This command cannot be used for secured programs. If there is no program in memory, executing this command causes an error.

Format



<Device name>........The name of the device to which the list of variables will be output is specified. For device names, refer to "1.4 Input/Output Control". When the device name is omitted, the display (CRT) is automatically specified. When one of the disk drives (A: through D:) is specified, a list file is created on the disk in the specified drive. The file name of the list file consists of the program name plus file type REF.

The list contains the following elements. A header is always output at the top of the list.



The output of a list can be aborted by depressing CANCEL or CTRL / C

Example 1

«Display»	«Key Operation and Explanation»
	• Output a list of the variables used in the program in memory on the display.
$_	[X][R][E][F][↵]

```
XYZ. 1204
A  10    20    30
B  10    80
C  10   100
$_
```

Example 2

«Display»	«Key Operation and Explanation»
	• Output a list of the variables used in the program in memory to the printer connected to connector 1.
$_	[X][R][E][F] [SPACE] [L][P][T][:][↵]

Printout

```
XYZ. 1204
A   10    20    30
B   10    80
C   10   100
```

$_

2.4.7 RUN Command

The RUN command activates the interpreter to start BASIC program execution. This command can also execute a program in the debugging mode by specifying /D.

Format

```
[RUN ␣] [<Program specification] [/D] [;<Character string>]
  ||
 [F7]
```

RUN .... The keyword can be omitted.

<Program specification> .... The name of the program that will be executed is specified. When the drive name is omitted, the current drive is automatically specified. The file type BAS may be omitted. When program specification is omitted, the priority program (program in memory) is automatically specified. The relationship between program specification and the program that will executed are summarized below.

a) When the program name is not specified:
   - The program in memory is executed.
   - If there is no program in memory, the command is not executed and the system returns to the command waiting state.

b) When the program name is specified:
   - The specified program is automatically loaded from the disk to memory and executed. In this case, the program already in memory is deleted.
   - If the specified program is not found on the disk, the message "<Program specification>NOT EXIST!" is displayed, the command is not executed, and the system returns to the command waiting state. In this case, the program in the memory is not deleted.

/D .... This is specified to execute the program in the debugging mode. For details, refer to "2.6 Debugging".

;<Character string> .... This specifies the character string that will be assigned to the COM$ function in a program. For details, refer to the explanation of the COM$ function given in "3. Built-in Function of Chapter III Language".

Example 1

«Display»	«Key Operation and Explanation»
	●The priority program is executed.
$_	
	[↵]
Program execution	

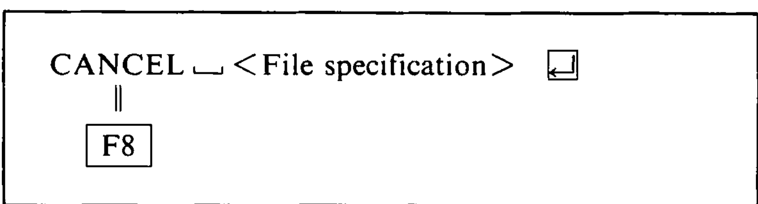
Example 2

«Display»	«Key Operation and Explanation»
	●Load program "ABC" from the disk in drive B to memory and execute it.
$_	
	[B][:][A][B][C][↵]
Program execution	

2.4.8 CANCEL Command

The CANCEL command deletes a specified program file or data file from the disk.

Format



<File specification> .... The file that will be deleted from the disk is specified. When the drive name is omitted, the current drive is automatically specified. When the file type is omitted, file type BAS (BASIC program file) is automatically specified.

The following confirmation message is displayed when this command is entered.

<File specification> `CANCEL (Y/N)?_`

When [Y] and [↵] are depressed, the specified program is deleted from the disk. When [N] and [↵] are depressed, the command is not executed and the system returns to the command waiting state.

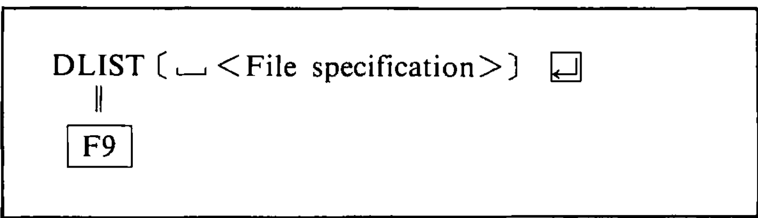
Example

«Display»	«Key Operation and Explanation»
	●Delete file "ABC.DAT" from the disk in drive B.
`$_`	
	[C][A][N][C][E][L] [SPACE] [B][:][A][B][C][.][D][A][T][↵]
`B:ABC.DAT CANCEL (Y/N)?_`	
	[Y][↵]
`$_`	

2.4.9 DLIST Command (Disk List)

The DLIST command outputs a list of specified files on a specified disk to the display. This command permits wild card specification so that more than one file can be specified at a time.

Format



<File specification> .... The names of the files that will be listed are specified. When only the drive name is specified, all of the files on the disk in the drive specified are automatically specified and all of their names are listed. When the drive name is omitted, the current drive is automatically specified. Wild card specification, the specification of more than one file at a time, is also possible.

Wild card specification

A wild card is used to specify more than one file at a time and is indicated by the generalization symbols "*" and "?". The symbol "*" is used to indicate all character strings. The symbol "?" is used to indicate all single characters.

The following are examples of wild card specification:

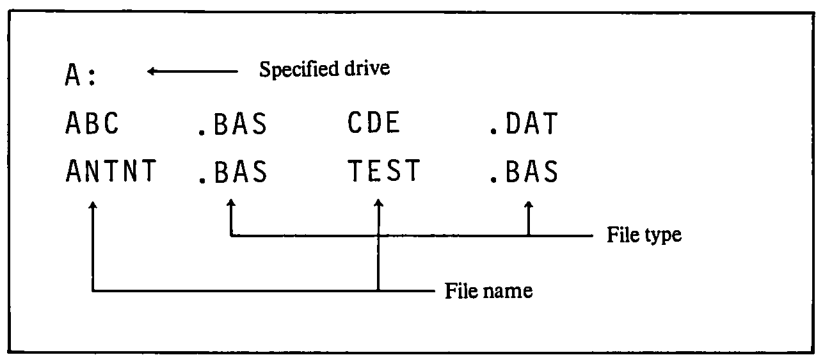
A: *.BAS ..... All BASIC program files on the disk in drive A.

B: A*.* ...... All files whose names begin with "A" on the disk in drive B (i.e. B:A.BAS and B:ABC.DAT).

B:???.* ...... All files with names of three letters on the disk in drive B (i.e. B:ABC.BAS, B:XYZ.DAT).

B:TEST?.* ... All files with names of five letters beginning with TEST (i.e. B:TESTA.BAS, B:TEST1.DAT, and B:TEST0.BAS).

The list of the files displayed by the DLIST command is as follows.



Example 1

«Display» «Key Operation and Explanation»

● Display a list of the BASIC program files on the disk in drive A.

```
$_
```

[D][L][I][S][T][SPACE][*][•][B][A][S][↵]

```
A:
ABC    .BAS    AAA    .BAS
OPQ    .BAS    TESTX  .BAS
   ~              ~
```

```
$_
```

Example 2

«Display» «Key Operation and Explanation»

● Confirm that program file "TEST.BAS" is on the disk in drive A.

```
$_
```

[D][L][I][S][T] [SPACE] [T][E][S][T][•][B][A][S][↵]

```
A:
TEST .BAS
```

← A name is not displayed if the specified file is not found on the disk.

```
$_
```

2.4.10 RNAME Command (Rename)

The RNAME command changes the name of a file on a disk.

Format

```
RNAME ␣ <File specification 1> ␣ TO ␣ <File specification 2> ↵
  ||
 [F10]
```

<File specification 1> ..... The file whose name will be changed is specified. When the drive name is omitted, the current drive is automatically specified. Wild card specification is not available.

<File specification 2> ..... The new file name is specified. The specification consists of <File name> and <File type>. Drive specification cannot be made. Wild card specification is not available.

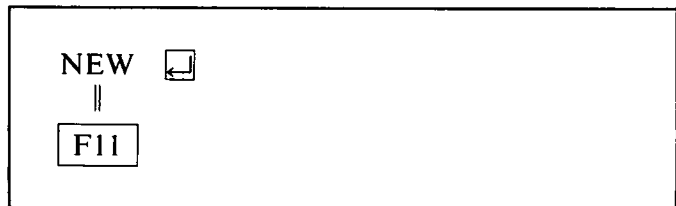
Example

«Display»	«Key Operation and Explanation»
	•Change the name of program file "ABC.BAS" on the disk in drive A to "XYZ.BAS."
$_	[R][N][A][M][E] [SPACE] [A][B][C][·][B][A][S] [SPACE] [T][O] [SPACE] [X][Y][Z][·][B][A][S][↵]
$_	

2.4.11 NEW Command

The NEW command deletes the program in memory and clears the display. Then the display returns to its original status at the time of BASIC start-up.

Format



Example

«Display» «Key Operation and Explanation»

```
$_
```

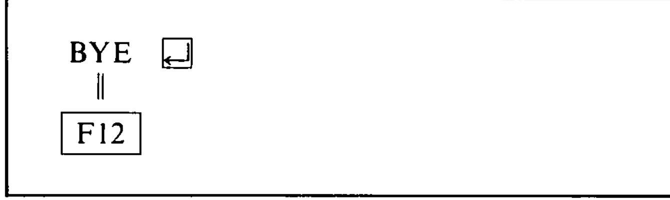
[N][E][W][↵]

```
Canon AS-100 BASIC Vm.mm
User's Memory nnnnnn Bytes
$_
```

2.4.12 BYE Command

The BYE command terminates BASIC and returns the system to the OS mode. The definitions made under BASIC return to their initial value (CP/M-86 initial state). At this time, the display is cleared and a cursor and prompting "A>" (indicating the OS mode) are displayed in the upper left-hand corner of the screen.

Format



Example

«Display» «Key Operation and Explanation»

```
$_
```

[B][Y][E][↵]

```
A>_
```

2.4.13 OS Mode Commands

In the OS mode, before BASIC is started-up or after BASIC is terminated using the BYE command, "A>" (A indicates the current drive) is displayed, and the AS-100 is put under the direct control of the CP/M-86 operating system.

In the OS mode, the commands supported by CP/M-86 can be used. Like commands under BASIC, they can be used with the AS-100 System.

This section explains the command functions of several OS mode commands which are useful to operate BASIC. Refer to the "CP/M-86 User's Manual" for OS commands not explained here and for the details of each command.

(1) VOLCOPY Command and COPYDISK Command

These commands copy the contents of one disk to another disk, thus preparing a copy (backup) of the disk. The COPYDISK command copies the disk by sector, and the VOLCOPY command by track. Their key operations are almost the same. The processing speed of the VOLCOPY command is slightly faster than that of the COPYDISK command. Before a disk copy is made using these commands, the disk on which the copy will be made must be initialized using the FORMAT command. The original disk (source) and copy disk (destination) must be the same size.

Format

{ VOLCOPY / COPYDISK } ↵

Operations are made as follows:

«Display» «Key Operation and Explanation»

A>_

[C][O][P][Y][D][I][S][K][↵]
(Or [V][O][L][C][O][P][Y][↵])

CP/M-86 Full Disk Copy Utility Version 2.0 or
VOLCOPY V1.01

Enter Source Disk Drive(A-D)?_

- Enter the name of the drive where the original disk is set.

[A][↵]

Destination Disk Drive(A-D)?_

- Enter the name of the drive where the copy disk is set.

[B][↵]

```
Copying disk A:to disk B:
             └ Source └ Destination

Is this what you want to do(Y/N)?_
```

- Confirm that the original disk and the copy disk are correct, and depress [Y] and [↵]. When [N] and [↵] are depressed, copying is not performed and the message "Copy another disk (Y/N)?" is displayed.

[Y][↵]

- The numbers of the tracks being processed are displayed in sequence. The displays for the COPYDISK command and the VOLCOPY command are slightly different.

```
Copy started
Reading track nn
Writing track nn
Verifying track nn
```

(COPYDISK only)

```
COPY TRACK NUMBER=nn
```

(VOLCOPY only)

```
Copy completed
```

(COPYDISK only)

```
"Copy another disk(Y/N)?"
```

- When copying is complete, this message is displayed. When [Y] and [↵] are depressed, the system returns to the beginning of command operation procedure and copying is repeated for the new disk. When [N] and [↵] are depressed, copying ends.

```
Copy program existing
```

(COPYDISK only)

```
A>_
```

(2) **FORMAT Command**

The FORMAT command initializes a disk.

Format

FORMAT [↵]

«Display» «Key Operation and Explanation»

```
A>_
```

[F][O][R][M][A][T][↵]

```
FORMAT Vn.nn
Disk B:will be destroyed,OK?_
```

●Set the disk that will be initialized in drive B and depress [Y] and [↵]. When [N] and [↵] are depressed, the disk is not initialized, and "A>_" is displayed.

```
COPYING SECONDARY BOOT.
```

```
A>_
```

●"A>_" is displayed when initialization is completed.

(3) **PIP Command**

The PIP command copies a file on a disk to another disk.

Format

PIP ␣ <Drive name> = <File specification> ↵

<Drive name> .... Specify the drive where the destination disk is set. The drive name cannot be omitted.

<File specification> ... Specify the original file by <Drive name>, <File name>, and <File type>. When the drive name is omitted, the current drive is automatically specified. Wild card specification can be made for the <File name> and <File type>.

«Display»	«Key Operation and Explanation»
A>_	•Copy program file "TEST.BAS" from the disk in drive A to the disk in drive B.
	[P][I][P][SPASE][B][:][=][T][E][S][T][.][B][A][S][↵]
A>_	

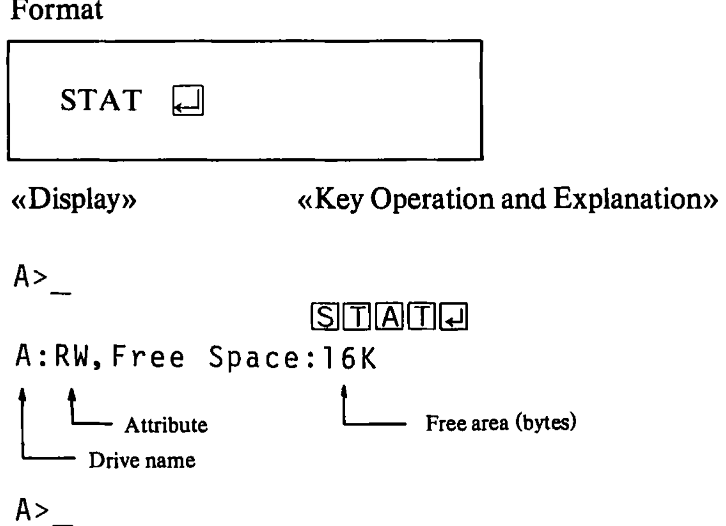
«Display»	«Key Operation and Explanation»
A>_	•Use wild card specification to copy all BASIC programs on the disk in drive A to the disk in drive B.
	[P][I][P][SPASE][B][:][=][*][.][B][A][S][↵]
Copying–	
TEST1.BAS	•The names of files copied are displayed when the wild card is specified.
⸾	
A>_	

If the destination disk contains a file of the same name as the file specified in PIP command , the file on the destination disk is automatically deleted and copying is performed. If the file on the destination disk is write-protected (see the CP/M-86 User's Manual), the message "DESTINATION IS R/O, DELETE (Y/N)?" is displayed. If [Y] is depressed, the file on the destination disk is deleted and the new file of the same name is copied following the message. If [N] is depressed, copying is not performed.

(4) **STAT Command**

The STAT command displays the size of the free area on a disk.

Format



```
STAT ↵
```

«Display» «Key Operation and Explanation»

```
A>_
                              [S][T][A][T][↵]
A:RW,Free Space:16K
```

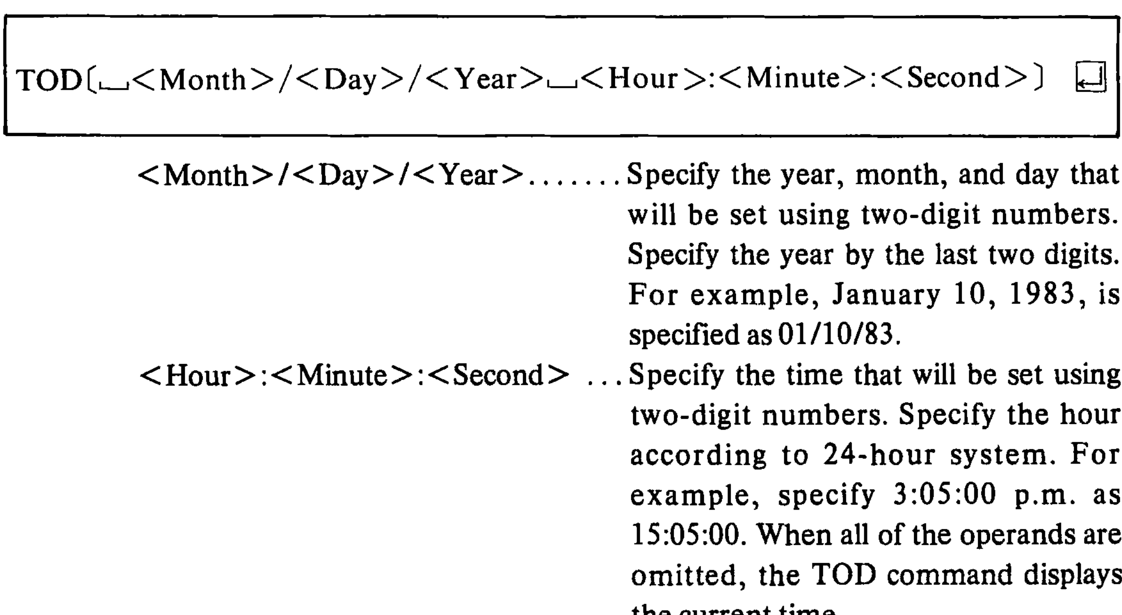


```
A>_
```

(5) **TOD Command**

The TOD command sets the AS-100's internal clock or displays its current value. In BASIC programs, the value of the internal clock can be modified or read using the TOD$ and TIM functions. The internal clock is set at 00 hours 00 minutes 00 seconds when the power is turned on. The elapsed time is recorded in seconds. Random values are set for date and day as the initial values. If the optional real-time clock is added, the elapsed time is counted even when the power is off.

Format



TOD[␣<Month>/<Day>/<Year>␣<Hour>:<Minute>:<Second>] ↵

<Month>/<Day>/<Year>....... Specify the year, month, and day that will be set using two-digit numbers. Specify the year by the last two digits. For example, January 10, 1983, is specified as 01/10/83.

<Hour>:<Minute>:<Second> ... Specify the time that will be set using two-digit numbers. Specify the hour according to 24-hour system. For example, specify 3:05:00 p.m. as 15:05:00. When all of the operands are omitted, the TOD command displays the current time.

«Display» «Operation and Explanation»

●Set the internal clock for 7:25 a.m., March 3, 1983.

```
A>_
```

[T][O][D] [SPASE] [0][3][/][0][3][/][8][3] [SPACE] [0][7][:][2][5][:][0][0][↵]

```
Strik key to set time
```

●The specified time is set when any key is depressed.

[↵]

```
03/03/83(Thu),07:25:00
```

●The time set is displayed. The day of the week is automatically set.

```
A>_
```

«Display» «Key Operation and Explanation»

●The current value of the internal clock is displayed.

```
A>_
```

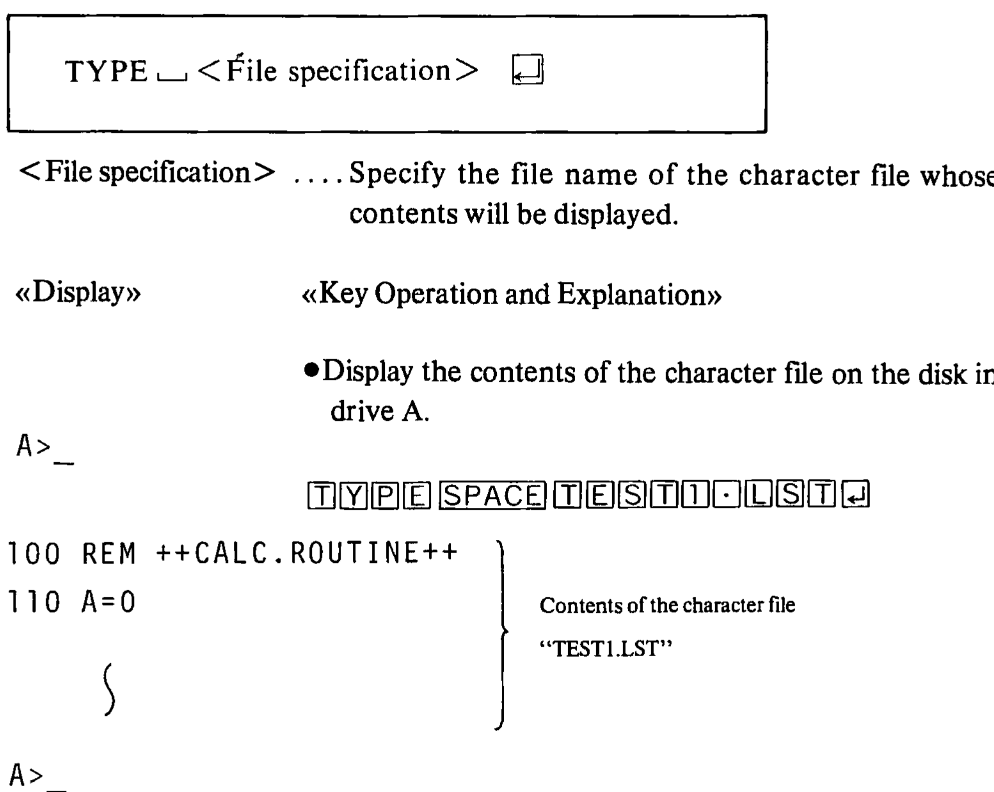
[T][O][D][↵]

```
03/03/83(Thu),07:26:05
A>_
```

(6) TYPE Command

The TYPE command displays the contents of a character file. For example, if this command is used when inputs are made from the character file using the G command during program editing (see "2.3.4 Program Editing"), the contents of the character file can be checked in advance.

Format



```
TYPE ␣ <File specification> ↵
```

<File specification> .... Specify the file name of the character file whose contents will be displayed.

«Display» «Key Operation and Explanation»

- Display the contents of the character file on the disk in drive A.

```
A>_
```

[T][Y][P][E] [SPACE] [T][E][S][T][1][.][L][S][T][↵]

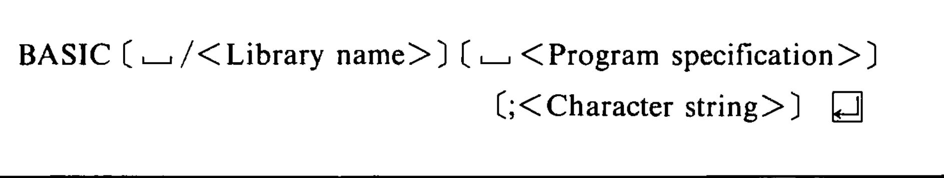
```
100 REM ++CALC.ROUTINE++
110 A=0
      ⋮
A>_
```

Contents of the character file "TEST1.LST"

(7) **BASIC Command**

The BAISC command activates the BASIC system. Specify the library names in this command when the library modules (ISAM and MATRIX libraries) will be loaded after the BASIC system module. This command can also specify the BASIC program which will be executed immediately after BASIC start-up.

Format



/<Library name> .... This is specified to load a library module in memory where it will reside following the BASIC system. A library is a file with file type LIB, which is required to use matrix-related instructions, etc. If the library name is not specified in the operand of the BASIC command, it must be loaded during the program execution to use the function. The benefits of having the library reside in memory are:

a) The memory is reserved for the BASIC system including the library, so the user's memory area displayed at BASIC start-up indicates the actual memory amount which can be used for user programs.
b) The processing speed improves because a library does not have to be loaded in a program.

<Program specification> .... Specify the BASIC program that will be executed just after BASIC start-up. The file type BAS may be omitted.

<Character string> .... This is the same as the character string of the RUN command of BASIC.

When the BASIC command that specifies the program that will be executed just after BASIC start-up is executed in the SUBMIT file (refer to the "CP/M-86 User's Manual"), the next line of the SUBMIT file is executed when the specified BASIC program ends with the BYE statement.

«Display» «Operation and Explanation»

- BASIC is activated and then the BASIC program "TEST" is executed. The MATRIX library resides in memory just after the BASIC system.

```
A>_
```

[B][A][S][I][C][SPACE][/][M][A][T][R][I][X] [SPACE][T][E][S][T][↵]

Version no.

```
Canon AS-100 BASIC Vn.nn
Copyright by Canon Inc.
Option:MATRIX
User's Memory mmmmmm Bytes
```

User area (byte)

◇ The BASIC program "TEST" is executed.

```
$_
```

2.4.14 Handlers

When using an optional printer with the AS-100, it is necessary to load handlers to memory before BASIC start-up (in the OS mode). The handlers and their loading operations are shown below. Refer to the "CP/M-86 User's Manual" for details of the handlers.

- A1200 . . . . . Load this to use the A-1200 Wire Dot Printer connected to connector 1.

  Operation: [A][1][2][0][0][↵]

- A1210 . . . . . Load this to use the A-1210 Color Printer connected to connector 1.

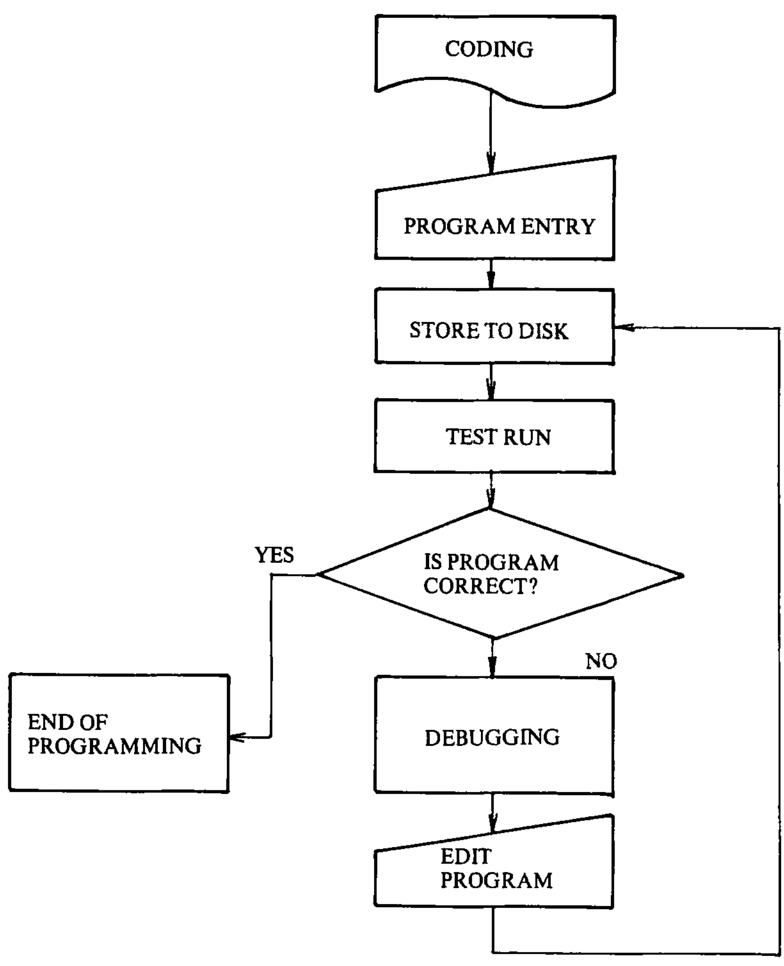
  Operation: [A][1][2][1][0][↵]

2.5 Programming

Creating a program is called programming. This section explains the programming procedure in sequence—from writing the program on a coding sheet and entering the program through the keyboard, to editing it and storing it to a disk. Refer to "2.6 Debugging" for details on debugging (finding errors) in the program.

2.5.1 Programming Procedure

The procedure that follows must be used to code the program or write it on the coding sheet, enter the program, and actually execute it.



2.5.2 Coding

First, write the program on a coding sheet. Since Canon BASIC programs accomodate up to 127 characters per line, including the line number and ↵ , use a 128-character coding sheet. Refer to "Chapter III Language" for the details of Canon BASIC language used for programming.

Coding Example:

Coding Sheet

Canon

PROGRAM NO.	0023	PROGRAM TITLE	TEST23	PROGRAMMER	Yoshizawa

LINE NO.	KEYWORD & OPERAND
10	REM +++ DATA CHECK PROGRAM +++
20	INTEGER NO,CODE,A,B,C
30	DIM A(20),B(20),C(20)
40	REM == INITIALIZE ROUTINE ==
50	PRINT "CHECK PROGRAM NO.1"
60	INPUT MSG("INPUT YOUR PASSWORD!")NAME$
70	OPEN #1,"A:PASS.DAT"
80	GET #1 WORK$:IF NAME$<>WORK$ THEN CLOSE #1:GOTO 60
90	CLOSE #1
100	REM == MENU PRINT ==

2.5.3 Program Entry

Enter the program through the keyboard according to the coding sheet. Use the following procedure to enter the program.

(1) Set the AS-100 in the programming mode.
(2) Enter each program line.
(3) Release the programming mode.
(4) Save the program to a disk.

(1) Setting the Programming Mode

Before the program is entered, the editor must be activated and the AS-100 must be set in the programming mode using the EDIT command.

Specify the program name in the operand of the EDIT command as shown below.

[E][D][I][T] [SPACE] <Program name> [↵]

The BASIC editor, in turn, activates the automatic numbering function, so the line number is displayed instead of "%".

```
PROGRAM CREATION xxxxxxxxx
                 └──────── Program name
10 ␣_
```

The program can be entered when this is displayed. The line number increases by 10 with the entry of each line.

Note: The program name can be omitted if a program is not loaded in memory. The program name must be defined by the N command (described later) before the programming mode is released.

(2) **Entering Program Lines**

After setting the AS-100 in the programming mode, enter the program lines according to the coding sheet. Depress [↵] at the end of each line. It is not necessary to enter line numbers because they are entered automatically. The program must not take up more than half the user memory area or execeed 32K bytes in length.

Example 1

«Display» «Key Operation and Explanation»

```
$_
                    [E][D][I][T] [SPACE] [A][A][A][↵]
                                          └─ program name
PROGRAM CREATION AAA
10_
                    [D][I][M] [SPACE] [A][(][1][0][)]
10 DIM A(10)_
                    [↵]
20_
```

Keywords of instructions can be entered by depressing the [F1] key and a typewriter key using the one-key, one-instruction function. For details of the one-key, one-instruction function, refer to "2.2.3, (5) One-key, one-instruction function".

If a mistake is made during program line entry, the line is not entered to the system even if [↵] is depressed. The cursor will automatically move to the position of the error. Then correct the error and depress [↵]. The line is then entered to the system and the next line number is displayed.

The following keys are used for correction.

[←] [→] : The cursor moves one character in the direction of the arrow. The functions are repeated if the keys are continuously depressed.

[INSERT] : This key inserts a character string at the cursor position. First depress [INSERT] and then enter the character string that will be inserted. Insertion is completed when the cursor is moved or when any key other than the character keys are depressed.

DELETE : This key deletes the character at the cursor position and shifts the following character string one column to the left.

→| |← : These keys move the cursor to the beginning of the line (|←) or to the end of the line (→|).

DEL : This key deletes the character immediately preceding the cursor and shifts the cursor one column to the left.

DELETE LINE : This key deletes the entire line except for the line number.

Example 2

«Display»	«Key Operation and Explanation»
a) 40 PRRINT A_ ⇧	● Various correction operations are shown.
	|← →→
40 PRRINT A	
	DELETE
40 PRINT A	
b) 80 LET B+C_ ⇧	
	←←←
80 LET B+C	
	INSERT A =
80 LET A=B+C	
c) 90 PRINT "ABC"_	
	←←←←
90 PRINT "ABC"	
	DELETE DELETE DELETE
90 PRINT " "	
	INSERT X Y Z
90 PRINT "XYZ"	

d)

```
100 GOTO 10000_
```

[DEL] [DEL]

```
100 GOTO 100_
```

e)

```
70 INPUT A_
```

[DELETE LINE]

```
70_
```

[SPACE] [P][R][I][N][T] [SPACE] [A]

```
70 PRINT A_
```

Programs can be entered using the basic operations described in this section. The details of the editor, including the procedure for using the edit commands, can be found in "2.5.4 Program Editing"

(3) **Releasing the Programming Mode**

After all program lines have been entered, release the programming mode using the following procedure:

1) Depress [↵] to display "%_".
2) Depress [E] and [↵] to release the programming mode and set the operating mode and then "$_" is displayed.

Now the program is in memory and can be executed. To prevent the program from being erased, save the program to a disk prior to execution.

(4) **Saving the Program to a Disk**

Use the SAVE command to save the entered program to a disk. The operand of the SAVE command can be omitted.

1) Enter [S][A][V][E][↵].
   When saving a program to a disk other than the one in drive A, specify the disk as follows:
   [S][A][V][E] [SPACE] <Drive name>: [↵]
2) The message "SAVE TO A: <Program name>.BAS(Y/N)?_" is displayed.
3) Depress [Y] and [↵].
4) The program in memory is saved to the disk in drive A. The system then returns to the command waiting state.

2.5.4 Program Editing

This section contains a detailed explanation of the editor functions. The operations just described use only a fraction of the editor functions. The editor is also used whenever an error is found during program checking or anytime a part of the program is modified. Program editing is just about the same as program creation. The only difference is whether the program being edited is in memory or not.

(1) Activating the Editor

Activate the editor using the EDIT command. Refer to "2.4.2 EDIT Command" for the details of this operation. The first 10 lines of the program are listed, and the programming mode prompting "%_" is displayed.

(2) Basic Editor Functions

The Canon BASIC editor is a line editor which means that it edits a program line by line.

There are two display statuses in the programming mode. One is the edit command waiting state when edit commands can be entered. In this state, prompting "%" is displayed. The other is the line input state, when program lines can be entered and modified directly. Either a line number or a program line with its line number is displayed.

Edit command waiting state:

```
%_
 ↑
 └── Cursor
```

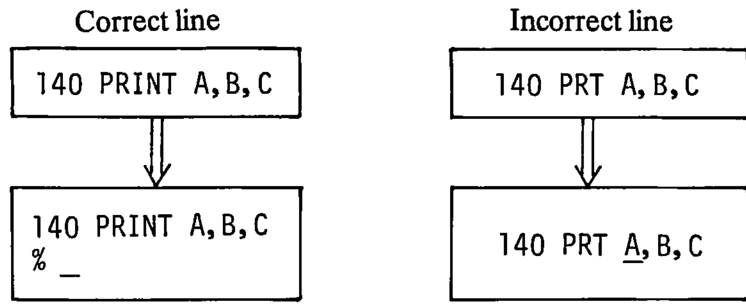
Line input state:

```
120_PRINT A,B,C
   ↑
   └── Cursor
```

One line in a program is defined as the key line for editing. This line is called the current line. The current line changes according to editing procedure. Basically the line displayed in the line input state is defined as the current line.

```
120 PRINT A,B,C
130 GOTO 100
140 [SUB1] REM ===
%C130
130_GOTO 100  ← Current line
```

The editor checks the grammar (syntax) of each line as it is entered and stores the lines as a program. If there is a syntax error in a line, the line is not entered to memory and the cursor will indicate the position of the error.



The syntax check performed by the editor starts at the beginning of a line. This means that the cursor will be positioned at the character immediately following the error.

(3) **Edit Commands**

Eleven edit commands can be used in the edit command waiting state of the programming mode. Their functions are described below.

1) LINE command.... Inserts, deletes, or modifies a line.
2) I command........ Specifies automatic numbering.
3) R command....... Renumbers lines.
4) L command .... Displays a list of programs.
5) D command....... Deletes a specified line.
6) C command .... Calls a specified line.
7) N command....... Changes a program name.
8) G command....... Reads a character file.
9) M command ...... Merges one BASIC program file with another one.
10) E command .... Releases the programming mode.
11) ↑↓ command ..... Changes a line.

1) **LINE Command**

The LINE command inserts, modifies, or deletes a specified line.

Format

```
<Line No.> [<Contents of line>] ↵
```

<Line No.> .... Specify the number of the line that will be inserted, modified, or deleted.

<Contents of line.> ... Specify the contents of the line. If this specification is omitted, the line is deleted.

The line specified becomes the current line. If the line is deleted, the line following it becomes the current line.

Example 1

«Display»	«Key Operation and Explanation»
	● Insert Line 125.
%_	
	[1][2][5] [SPACE] [A][=][A][+][B][↵]
%125 A=A+B	
%_	* If there is already a Line 125, its contents are changed.

Example 2

«Display»	«Key Operation and Explanation»
	● Delete Line 400.
%_	
	[4][0][0][↵]
%400	
%_	

2) **I Command (Input)**

This command specifies automatic numbering. The I command also sets the display in the line input state.

Format

```
I [<Starting line>] [, <Interval>]
```

<Starting line> ..... Specify the line number where auto numbering will start. If the starting line is omitted, auto numbering starts from the line with a number equal to the number of the last program line of the program plus the value specified in <Interval>.

For example, if the starting line is omitted during the editing of a program in which Line 500 is the last line and the I command specifying a numbering interval of 5 is executed, auto numbering starts from Line 505 with the subsequent lines automatically numbered 510, 515, 520. . . etc.

When a line already stored in the program is specified, auto numbering starts from the line with line number equal to the number of the line specified plus the value specified in <Interval>. But if that value (specified line number + interval) corresponds to a line already entered in the program, the I command is ignored. It is possible to specify F for the first line of the program and L for the last line.

<Interval>........ The automatic numbering interval is specified. If the interval is omitted, 10 is automatically specified.

When the line number of line input state set by the automatic numbering function matches or exceeds the line already entered in the program, auto numbering is released and the editor returns to the edit command waiting state. The last line entered immediately before line input state ends becomes the current line.

The line input state is also released when ↵ is depressed without entering the contents of a line in the line input state set by the automatic numbering function.

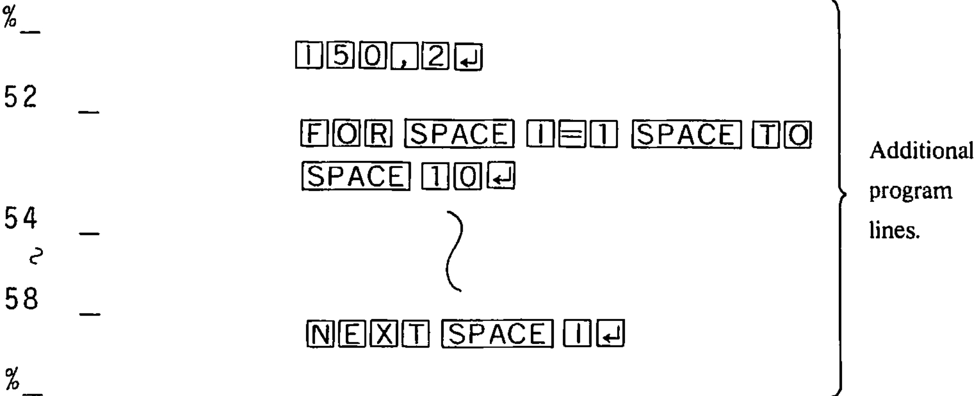
Example 1

«Display»	«Key Operation and Explanation»
	● Add Line 510 and subsequent lines to a program consisting of Lines 10～500.
%_	
	[I] [↵]
510 _	
	[A] [=] [B] [+] [C] [↵]
520 _	
≀	≀
600 _	
	[↵]
%_	

Example 2

«Display» «Key Operation and Explanation»

● Insert program lines at intervals of 2 between Lines 50 and 60.



3) R Command (Renumbering)

The R command renumbers the lines of a program at equal intervals. The lines specified in a statement like the GOTO statement are automatically renumbered.

Format

```
R [<Starting line>] [, <Interval>] ↵
```

<Starting line> .... Specify what the beginning line number of a program will be after renumbering. If this specification is omitted, the value specified in <Interval> is automatically specified as the first line of the program.

<Interval>........ Specify the renumbering interval for the line numbers. If this specification is omitted, 10 is automatically specified.

After this command is executed, the first 10 renumbered lines of the program are displayed. The last line displayed becomes the current line.

Example

«Display»	«Key Operation and Explanation»
	• Renumber lines at intervals of 100 beginning from Line 100.
%_	
	[R][,][1][0][0][↵]
100 INTEGER A, B, C	
200 DIM A(20),B(20),C(20)	
~	
1000 PRINT "OK?"	
%_	

4) **L Command (List)**

The L command displays a program list. The part that will be listed can be specified.

Format

```
L [ <Starting line>] [, <Ending line>] ↵
```

<Starting line> .... Specify the first line that will be listed. If the line specified is not found in the program, listing starts from the first line beyond the specified line. For example, if Line 25 is specified in a program containing Lines 10, 20, and 30, Line 30 becomes the starting line.

<Ending line>..... Specify the last line that will be listed. If this specification is omitted, 10 lines starting from the line specified by <Starting line> are automatically listed.

If both of the operands are omitted, 10 lines starting from the current line are listed. It is possible to specify F for the first line and L for the last line.

If the part of the program specified is not found, the message "LINE NOT EXIST!" is displayed.

The last line listed after execution of this command becomes the current line. The listing can be aborted using the [CANCEL] key or [CTRL] / [C] .

Example 1

«Display» «Key Operation and Explanation»

«Display»	«Key Operation and Explanation»
	• List Lines 100—400 of the program.
%_	
	[L][1][0][0][,][4][0][0][↵]
100—	
?	
400—	
%_	

Example 2

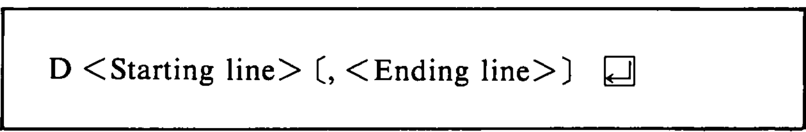
«Display»	«Key Operation and Explanation»
	• List the program from the first line to the last line.
%_	
	[L][F][,][L][↵]
10—	
20—	
?	
%_	

Note: If the lines listed exceed the display capacity, the lines displayed will be scrolled up and off the display as listing progresses. To stop scrolling, use the screen stop function described in "2.7 Functions of Control Key".

5) **D Command (Delete)**

The D command deletes a specified line.

Format


D <Starting line> [, <Ending line>] ↵

<Starting line>......... Specify the first line of the program that will be deleted. If the starting line is specified without specifying the ending line, only the specified starting line is deleted.

<Endling line>......... Specify the last line of the program that will be deleted.

When deleting only one line, if the specified line is not found in the program, the message "LINE NOT EXIST!" is displayed. When deleting more than one line at a time, if line specified as the starting line or the ending line is not found in the program, the lines that are found within the range specified are deleted. If none of the lines specified are found in the program, the message "LINE NOT EXIST!" is displayed. It is possible to specify F for the first line and L for the last line. The line immediately following the line deleted after this command is executed becomes the current line.

Example 1

«Display»	«Key Operation and Explanation»
	• Delete Line 120.
%_	
	D 1 2 0 ↵
%_	(Operation 1 2 0 ↵ is also acceptable.)

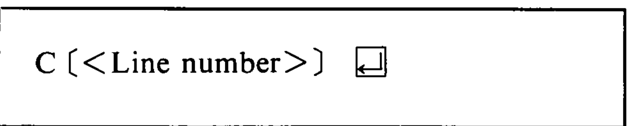
Example 2

«Display»	«Key Operation and Explanation»
	• Delete all lines from Line 150 through the last line of the program.
%_	
	D 1 5 0 , L ↵
%_	

6) **C Command (Call)**

The C command calls a specified line and sets the line input state.

Format



<Line number>........ Specify the program that will be called for correction. If this specification is omitted, the current line is automatically called.

When a program line is called using this command, its contents are displayed and the editor enters the line input state. Correct the contents and depress ↵ to enter the correction. When correction is complete, the editor automatically returns to the edit command waiting state.

After this command is entered, the line called becomes the current line.

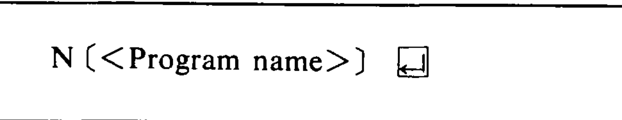
Example 1

«Display»	«Key Operation and Explanation»
	• Call and correct Line 150.
%_	
	[C][1][5][0][↵]
150_PRINT A	
	[→|][←][B]
150 PRINT B_	
	[↵]
%_	

7) **N Command (Name)**

The N command changes the name of a program.

Format


N [<Program name>] ↵

<Program name> ...... Specify the new program name. The program name must be a character string consisting of capital alphabet letters and numbers, beginning with a capital letter and not exceeding 8 characters.

When this command is executed, the old program name is displayed in the format "OLD PROGRAM NAME <Program name>". When the program name is omitted, the program name does not change and the current program name is displayed.

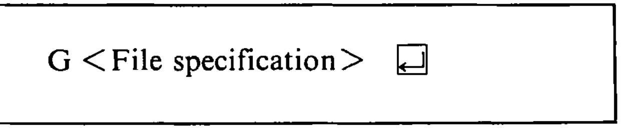
Example 1

«Display»	«Key Operation and Explanation»
	• Change the name of program "AAA" to "BBB".
%_	
	N B B B ↵
OLD PROGRAM NAME AAA	
%_	

8) **G (Get) Command**

The G command reads a character string from a character file on a disk. The character string is treated the same as program entry through the keyboard and a syntax check is performed.

Format



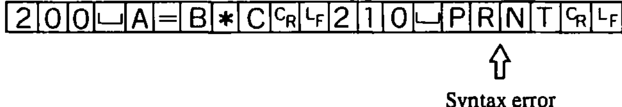
<File specification> .... Specify the character file that will be read. If the specified file is not found on the disk, the message "<File specification> NOT FOUND!" is displayed and the editor enters the edit command waiting state.

A character string which is read from a character file using this command is entered to the editor. If a syntax error is found in the character string which is read from the file, reading is stopped temporarily and the line is displayed with the cursor positioned at the error. When the error is corrected and ↵ is depressed, reading is resumed.

Example

«Display» «Key Operation and Explanation»

- A character file "ABC.LST" with the following contents is read and entered:




```
%_
```

G A : A B C . L S T ↵

```
200 A=B*C
210 PRNT_
```

... Suspended due to to the syntax error

← ← INSERT I ... Correction

```
210 PRINT
```

↵ .... Correction completed

```
%_
```

- Line 200 and 210 are entered after the operation above.

Note: Program line entry from a character file is treated the same as an entry through the keyboard. So, if a program line whose line number has already been used in the program in the memory is specified with program line entry from a character file, the program line in memory is updated to the contents specified by the character string read from the character file.

9) **M Command (Merge)**

The M command merges a part of a BASIC program with another program. The range of program lines that will be merged can be specified. Only BASIC programs created by this editor and consisting of an intermediate code can be merged. Secured programs cannot be merged.

Format

M <Program specification> [<Starting line> [, <Ending line>]] ↵

<Program specification> .... Specify the program that will be merged. When the drive name is omitted, the current drive is automatically specified. File type BAS may be omitted.
If the specified program is not found, the message "<Program name> NOT FOUND! " is displayed and the editor returns to the edit command waiting state.

<Starting line>[, <Ending line>] ...... Specify the part of the program that will be merged by line numbers. The method of specification is similar to that of the D command. When the merge range specification is omitted, all program lines are merged.

Example

«Display»	«Key Operation and Explanation»
	● Merge lines 100—200 of program "XYZ" on the disk in drive A with the program in memory.
%_	
	[M][X][Y][Z][1][0][0][,][2][0][0][↵]
	● The merged program lines are listed.
%_	
	[L][1][0][0][,][2][0][0][↵]
100 ——	
~	
200 ——	
%_	

10) **E Command (End)**

The E command releases the programming mode and sets the system in the operating mode.

Format

```
E ↵
```

«Display»	«Key Operation and Explanation»
%_	
	[E][↵]
$_	

11) **↑↓ Command (Up/Down)**

This command is executed by the cursor control keys [↑] and [↓]. This operation displays the line immediately preceding ([↑]) or following ([↓]) the line currently displayed and sets the line input state. A warning buzzer sounds if [↑] is depressed at the beginning of a program or [↓] is depressed at the end of a program.

Format

[↑] or [↓]

«Display»	«Key Operation and Explanation»
120 _PRINT A	
	[↑]
110 _A=A+1	
	[→][B]
110 B=A+1	
	[↵]
%_	

(

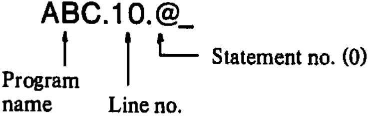
2.6 Debugging

Debugging is finding the incorrect parts of a program and correcting them. When the RUN command in which "/D" is specified is executed, the program is executed in the debugging mode and program can be checked during execution. Program errors that are found are corrected using the editor.

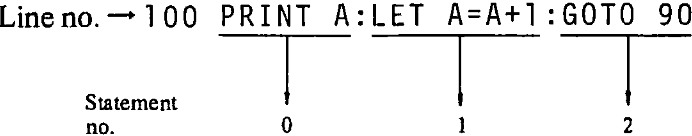
2.6.1 Debugging Mode Outline

In the debugging mode, the program can be executed statement by statement. That is, the instructions in the program are executed one by one so that the execution procedure of the program can be checked.

When program execution starts in the debugging mode, program exectuion is suspended and the debugging prompting "@" is displayed together with the program name, line number, and statement number as shown below, immediately before the first statement is executed.



Numbers (0, 1, 2, . . .) are assigned to statements (multistatements) written on the same line. As shown in the above example, however, statement 0 is not displayed.



This temporary suspension is called the debugging command waiting state. The following seven debugging commands can be entered.

1) R command .... Restarts program execution.
2) S command .... Executes one statement.
3) T command .... Specifies the section that will be traced.
4) B command .... Sets a breakpoint.
5) U command .... Releases a breakpoint.
6) D command .... Displays the contents of a variable.
7) E command .... Ends the program.

Trace is program execution in which the line and statement number of the statement currently being executed is displayed. The execution results of each statement can be checked one by one.

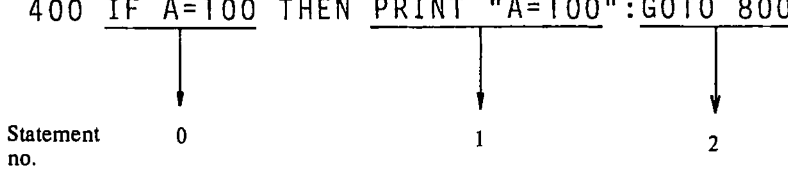
A breakpoint is a point set in a program where program execution is temporarily suspended. In the debugging mode, program execution is temporarily suspended just prior to the execution of a statement set as a breakpoint, and the system enters in the debugging command waiting state.

For example, the result of a calculation by a program is different than expected. If breakpoints are set at several points in the calculation routine and the value of the variable is checked at these points, the statement responsible for the incorrect calculation result can be found easily.

Even if the execution of a program is started without specifying "/D", program execution can be changed to debugging mode execution by depressing [CTRL]/[A].

When the [CTRL]/[A] keys are depressed, program execution is temporarily suspended and the system enters the debugging command waiting state.

Note: When the subkeyword THEN is included in the IF statement, the statement is treated as two statements like the example below.



2.6.2 Debugging Commands

(1) R Command (Run)

The R command restarts program execution starting from the statement being displayed (i.e. the statement indicated by the line number and statement number immediately preceding the prompting). When a breakpoint is set in the program, the program stops and the system enters the debugging command waiting state just prior to statement execution. When the specified trace section is executed, the line and statement numbers are displayed one by one during execution.

Format

```
R ↵
```

(2) S Command (Step)

The S command executes a statement being displayed (i.e. the statement indicated by the line number and statement number, immediately preceding the prompting). The system enters the debugging command waiting state after one statement is executed.

Format

```
↵
```

Note: The S command has no keyword. Simply depressing the ↵ key executes the S command.

(3) T Command (Trace)

The T command specifies a section for trace execution, in which a program is executed and the line and statement number of each statement are displayed as they are executed. The T command only specifies a trace section; actual trace execution is performed when program execution is restarted and the statements in the trace section are executed. A trace section can be specified in the subprogram called using the CALL statement.

Only one trace section can be specified in each program. If more than one trace section is specified, only the last trace section specified is valid.

Format

```
T [[<Program name>] <Starting line> [, <Ending line>] ↵
```

<Program name> ...... When the trace section is specified in the subprogram that will be called by the CALL statement, specify the subprogram name. If this specification is omitted, the trace section is set to the program whose execution is suspended temporarily. Therefore, when the system is in the debugging command waiting state in a subprogram, the name of the main program must be specified even if the trace section is set in the main program.
The program name must be specified with capital alphabet letters and numbers.

<Starting line> [,<Ending line>]. . . .
Specify a trace section by line number(s) as follows:

a) When only the starting line is specified:
Only that line is set to a trace section. If the line specified is not found in the program, the specification is valid but tracing is not actually performed.

b) When both the starting line and ending line are specified:
The section from the first statement on the line specified <Starting line> through the last statement on the line specified <Ending line> is trace section. If a line number specified in a program, those lines actually included in the specified section are traced. The relationship between the starting line and ending line are shown below.

Starting line < Ending line . . . Section specified
Starting line = Ending line . . . The line only
Starting line > Ending line . . . Error

c) When the trace section specification is omitted:
The trace section currently set is displayed as follows:
<Program name>.<Starting line>,<Ending line>

d) When a negative line number is specified:
The trace section specification is reset.

Note: If an error is caused by the T command with an incorrect operand, the trace section currently set is released. An effecive trace range is limited to one program. For example, even if a section containing the CALL statement is specified as a trace section, tracing is not performed in the subprogram called by the CALL statement.

(4) B Command (Break)

The B command sets a breakpoint at the line number specified. One breakpoint can be set with each B command. Up to four breakpoints can be set in a program. Program execution in the debugging mode is suspended immediately before executing a statement set as a breakpoint and the system enters the debugging command waiting state. The B command can also be used to set a breakpoint in a subprogram.

Format

```
B [[<Program name>.] <Line No.> [.<Statement No.>]] ↵
```

Program name .... Specify the name of the subprogram when a breakpoint is set to the subprogram that will be called by the CALL statement. For details, refer to the explanation of the T command.

<Line No.> [.<Statement No.>] .. Specify the numbers of the line and statement where the breakpoint will be set. If the statement number is omitted, 0 is automatically specified.

When the B command is executed to set a breakpoint when four breakpoints have already been set, the message "ALREADY 4 BREAK POINTS EXIST!" is displayed together with the line and statement numbers of the four breakpoints already set.

If all operands are omitted, the program name, line numbers, and statement numbers in which breakpoints are set are all displayed. If there are no breakpoints set, the message "NO BREAK POINT!" is displayed.

(5) **U Command (Unbreak)**

The U command releases a breakpoint.

Format

```
U [[<Program name>.] <Line No.> [.<Statement No.>]] ⏎
```

<Program name> ....Specify the name of the program in which the breakpoint will be released. For details, refer to the explanation of the B command.

<Line No.> [.<Statement No.>] .. Specify the number of the line and statement of the breakpoint that will be released. If the specified breakpoint is not set in the program, the message "NOT EXIST!" is displayed together with all breakpoints currently set. If all operands are omitted, all of the breakpoints are released. If breakpoints are not set, the message "NO BREAK POINT!" is displayed.

(6) **D Command (Display)**

The D command displays the current value of a variable.

Format

```
D <Variable name> ↵
```

<Variable name> ...... Specify the name of the variable whose current value will be displayed. When a varible specify its name with the subscript.
The display format is as follows:
- Arithmetic variables
  Same as the PRINT statement without a format.
- String type variables
  X"<Hexadecimal code>", "character string"

When a specified variable is not defined in a program, the message "VARIABLE NAME NOT FOUND!" is displayed.

(7) **E Command (End)**

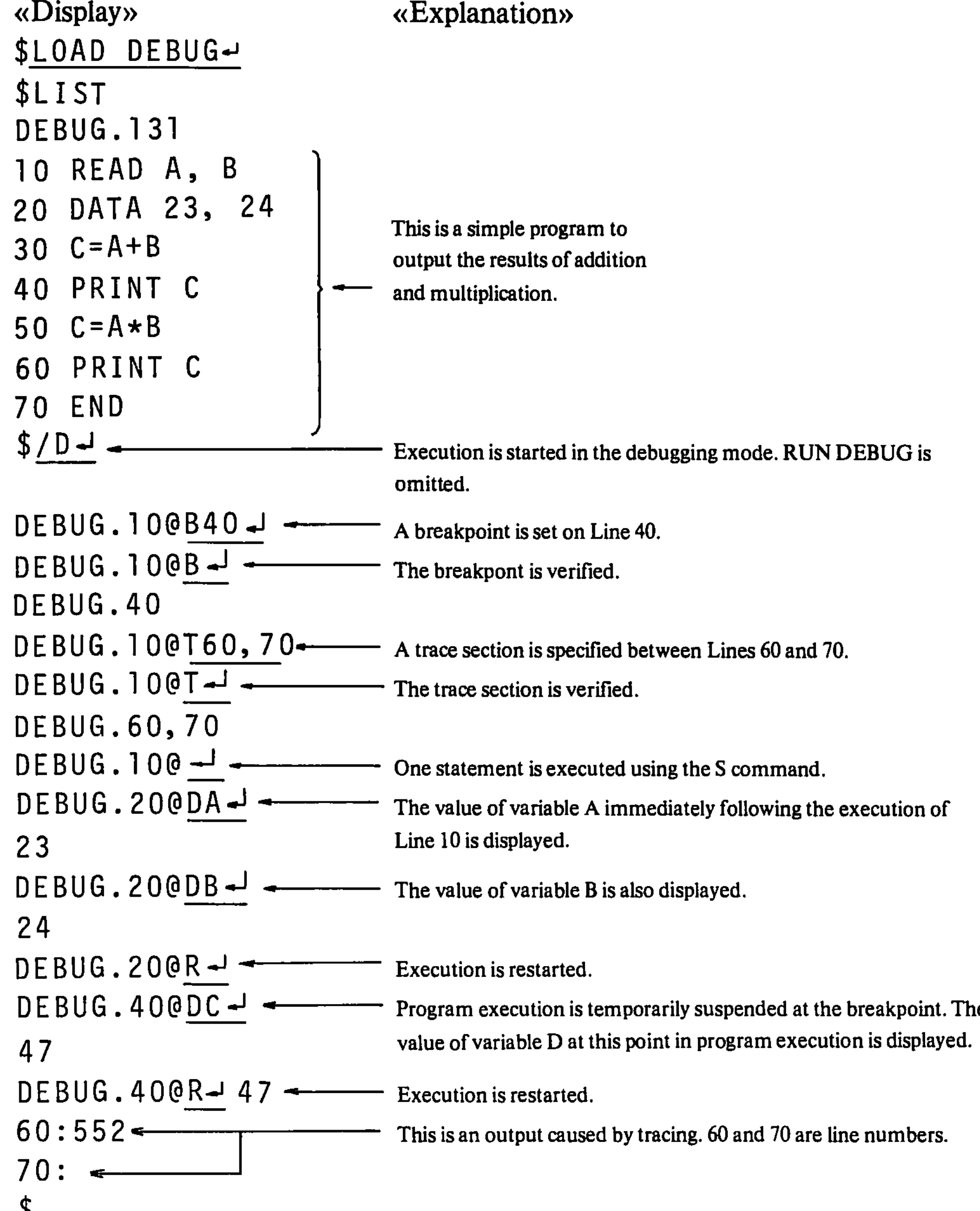
The E command ends program execution. The E command has the same function as the END statement.

Format

```
E ↵
```

2.6.3 Debugging Example

This section gives a step-by-step explanation of a simple program debugging example. Underlined portions of the display indicate inputs through the keyboard. ↵ indicates the ↵ key.



«Display» «Explanation»

```
$LOAD DEBUG↵
$LIST
DEBUG.131
10 READ A, B
20 DATA 23, 24
30 C=A+B
40 PRINT C              ← This is a simple program to output the results of addition and multiplication.
50 C=A*B
60 PRINT C
70 END
$/D↵                    ← Execution is started in the debugging mode. RUN DEBUG is omitted.
DEBUG.10@B40↵           ← A breakpoint is set on Line 40.
DEBUG.10@B↵             ← The breakpont is verified.
DEBUG.40
DEBUG.10@T60,70         ← A trace section is specified between Lines 60 and 70.
DEBUG.10@T↵             ← The trace section is verified.
DEBUG.60,70
DEBUG.10@↵              ← One statement is executed using the S command.
DEBUG.20@DA↵            ← The value of variable A immediately following the execution of Line 10 is displayed.
23
DEBUG.20@DB↵            ← The value of variable B is also displayed.
24
DEBUG.20@R↵             ← Execution is restarted.
DEBUG.40@DC↵            ← Program execution is temporarily suspended at the breakpoint. The value of variable D at this point in program execution is displayed.
47
DEBUG.40@R↵ 47          ← Execution is restarted.
60:552                  ← This is an output caused by tracing. 60 and 70 are line numbers.
70:
$_
```

2.7 Functions of Control Key

Several functions can be performed by simultaneously depressing [CTRL] and one of several predetermined keys. These are called console control operations, whose and their functions are described below.

- Abort Program Execution

Operation: [CTRL]/[C]

Function: Aborts program exection. Like [CANCEL], this operation is also valid to end of the output of lists, etc.

- Hard Copy of the Display

Operation: [CTRL]/[P]

Function: Outputs the display contents to the printer. The printer handler must be loaded in memory to use this function.

- Temporary Suspension of Output to the Display

Operation: [CTRL]/[S]

Function: Suspends output to the display.
When output on the screen exceeds the display capacity, this operation stops scrolling to permit confirmation.

- Restart Output to the Display

Operation: [CTRL] / [Q]

Function: Restarts output to the display after temporary suspension by [CTRL] / [S] .

- Break for Debugging

Operation: [CTRL] / [A]

Function: Sets the system in the debugging command waiting state during program execution. (See "2.6 Debugging").

- Switch Between Smooth Scroll and Line Scroll

Operation: [CTRL] / [2]

Function: Switches the display from line scroll to smooth scroll, or vice versa.

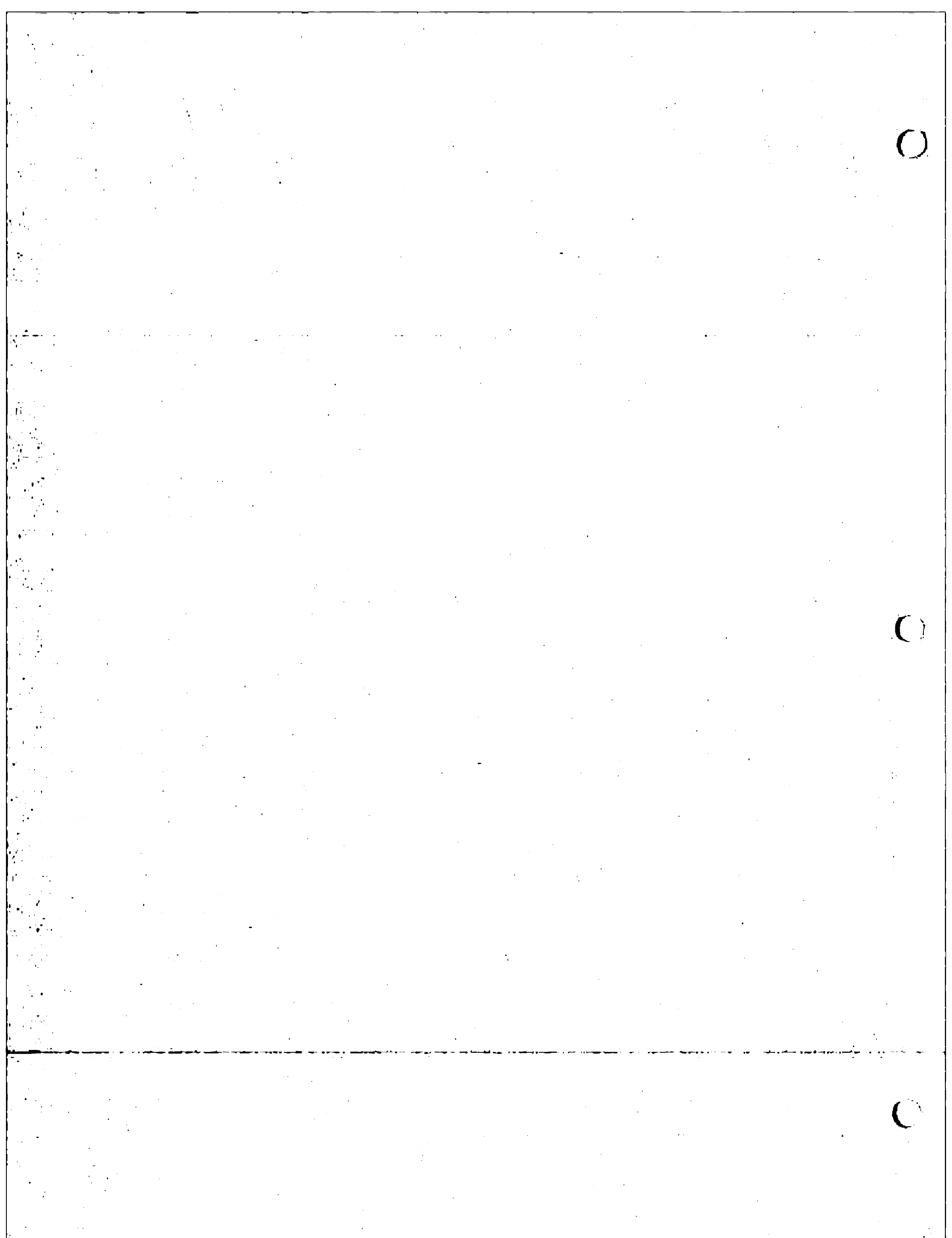
- Elimination of Click Tone

Operation: [CTRL] / [1]

Function: Eliminates the click sound that confirms key operations. The sound is restored by depressing [CTRL] / [1] again.

Chapter III — Language

Canon BASIC



This chapter explains the detailed specifications of Canon BASIC language.

Contents

1. **Program Elements** 1
   - 1.1 Program Lines 1
   - 1.2 Constants 2
   - 1.3 Variables 4
   - 1.4 Array Variables 5
   - 1.5 Arithmetic Operators 6
   - 1.6 Relational Operators and Expressions 7
   - 1.7 Logical Operators and Expressions 8
   - 1.8 Arithmetic Expressions 13
   - 1.9 String Expressions 14
   - 1.10 Files 14
   - 1.11 Logical Device Numbers 15

2. **Instructions** 16
   - 2.1 Formats 17
     - 2.1.1 Program Instruction Elements 17
     - 2.1.2 Symbols Used in Formats 17
     - 2.1.3 Format Interpretation Examples 18
   - 2.2 Declaration Instructions 19
     - 2.2.1 REM Statement 19
     - 2.2.2 DIM Statement 21
     - 2.2.3 OPTION BASE Statement 24
     - 2.2.4 INTEGER Statement 25
     - 2.2.5 DEFKEY Statement 27
   - 2.3 Assignment Instruction 29
     - 2.3.1 LET Statement 29
   - 2.4 Input Instructions 32
     - 2.4.1 INPUT Statement 32
     - 2.4.2 INPUT USING Statement 40
   - 2.5 Output Instructions 44
     - 2.5.1 PRINT Statement 44
     - 2.5.2 PRINT USING Statement
FORMAT Statement 58
   - 2.6 Branch Instructions 66
     - 2.6.1 GOTO Statement 66
     - 2.6.2 GOSUB Statement
RETURN Statement 67
     - 2.6.3 IF Statement 70
     - 2.6.4 ON Statement 77

2.7 Loop Instructions 80
2.7.1 FOR Statement
NEXT Statement 80
2.8 Constant Definition Instructions 85
2.8.1 READ Statement
DATA Statement 85
2.8.2 RESTORE Statement 88
2.9 Program Control Instructions 89
2.9.1 END Statement 89
2.9.2 BYE Statement 90
2.10 Function Definition Instruction 91
2.10.1 DEF FN Statement 91
2.11 Program Calling Instructions 93
2.11.1 CALL Statement
PARAM Statement 93
2.11.2 FREE Statement 98
2.12 File-Related Instructions 101
2.12.1 OPEN Statement 101
2.12.2 CLOSE Statement 105
2.12.3 CHANGE Statement 107
2.12.4 PUT Statement 109
2.12.5 GET Statement 119
2.12.6 Other Inputs/Outputs 126
2.13 Matrix-Related Instructions 131
2.13.1 Before Using Matric Related Instructions 131
2.13.2 Notes to Use of Matrix Related Instructions 137
2.13.3 MAT INPUT Statement 137
2.13.4 MAT READ Statement 139
2.13.5 MAT PRINT Statement 140
2.13.6 MAT MOV Statement 142
2.13.7 MAT ADD Statement 143
2.13.8 MAT SUB Statement 144
2.13.9 MAT MUL Statement 145
2.13.10 MAT DIV Statement 147
2.13.11 MAT SUM Statement 148
2.13.12 MAT CSUM Statement 149
2.13.13 MAT RSUM Statement 150
2.13.14 MAT IDN Statement 151
2.13.15 MAT INV Statement 152
2.13.16 MAT TRN Statement 153
2.13.17 MAT DET Statement 154
2.13.18 MAT MLT Statement 155
2.13.19 MAT MAX Statement 156
2.13.20 MAT MIN Statement 157

3. **Built-in Functions** 159
3.1 Arithmetic Functions 159
3.1.1 EXP Function 159
3.1.2 LOG Function 159
3.1.3 LGT Function 160
3.1.4 SQR Function 161
3.1.5 FRC Function 161
3.1.6 RND Function 162
3.1.7 ABS Function 163
3.1.8 SGN Function 163
3.1.9 FIX0 Function 164
3.1.10 FIX5 Function 165
3.1.11 FIX9 Function 166
3.1.12 FIXE Function 166
3.1.13 INT Function 167
3.1.14 SIN Function 168
3.1.15 COS Function 168
3.1.16 TAN Function 169
3.1.17 ASN Function 170
3.1.18 ACS Function 170
3.1.19 ATN Function 171
3.1.20 RAD Function 171
3.1.21 DMS Function 172
3.1.22 ARD Function 173
3.1.23 ADS Function 173
3.1.24 MOD Function 174
3.1.25 MAX Function 175
3.1.26 MIN Function 176
3.1.27 TIM Function 176
3.1.28 PI Function 177
3.1.29 SIZE Function 178
3.1.30 ERR Function 178
3.1.31 EOF Function 179
3.1.32 % CURX Function 181
3.1.33 % CURY Function 181
3.2 String Functions 183
3.2.1 LEN Function 183
3.2.2 IDX Function 184
3.2.3 VER Function 185
3.2.4 NUM Function 186
3.2.5 CHR$ Function 187
3.2.6 ASC$ Function 188
3.2.7 COD Function 189
3.2.8 STR$ Function 190

3.2.9 INPUT$ Funtion 193
3.2.10 KEY Function 195
3.2.11 FKEY Function 198
3.2.12 COM$ Function 201
3.2.13 HEX$ Function 202
3.2.14 TOD$ Function 203

4. ISAM Function 206
4.1 What Is ISAM 206
4.1.1 Indexed Access 206
4.1.2 Keys 208
4.2 Canon BASIC ISAM Function 209
4.2.1 General 209
4.2.2 Records 210
4.2.3 Primary Keys and Alternate Keys 211
4.2.4 Files 213
4.2.5 Pointer 213
4.2.6 Limitations and Notes for Use 215
4.3 How to Use ISAM Instructions 216
4.3.1 Loading the ISAM Library 216
4.3.2 Design of Files 216
4.3.3 Variables 217
4.3.4 Return Code 218
4.3.5 How to Interpret Formats 218
4.4 Basic ISAM Instructions 219
4.4.1 ISAM OPEN Statement 219
4.4.2 ISAM CLOSE Statement 224
4.5 ISAM Data Write Instructions 225
4.5.1 ISAM PACK Statement 225
4.5.2 ISAM WRITE Statement 228
4.5.3 ISAM REWRITE Statement 231
4.6 ISAM Data Read Instructions 235
4.6.1 ISAM UNPACK Statement 235
4.6.2 ISAM RREAD Statement 236
4.6.3 ISAM START Statement 240
4.6.4 ISAM SREAD Statement 242
4.7 Other ISAM Instructions 246
4.7.1 ISAM DELETE Statement 246
4.7.2 ISAM SECUR Statement 248
4.8 Return Code 250
4.9 ISAM Utility Programs 253
4.9.1 ISGEN Utility 253
4.9.2 IDXINF Utility 259
4.10 How To Calculate File Size 261

5. **Graphic Functions** 263
5.1 Graphic Functions 263
5.1.1 Coordinates 263
5.1.2 Palette and Display Color Specification 265
5.1.3 Current Point 268
5.1.4 Line Types 269
5.1.5 Pattern 269
5.1.6 How to use Graphic Instructions 270
5.2 Graphic Declaration Instructions 272
5.2.1 DEFCOL Statement 272
5.2.2 COLOR Statement 273
5.2.3 ORIGIN Statement 275
5.3 Graphic Drawing Instructions 277
5.3.1 PSET Statement 277
5.3.2 LINE Statement 278
5.3.3 RECT Statement 280
5.3.4 CIRCLE Statement 281
5.3.5 FAN Statement 283
5.3.6 ELLIP Statement 284
5.3.7 MARK Statement 285
5.4 Other Graphic Instructions 287
5.4.1 TEXT Statement 287
5.4.2 PAINT Statement 288
5.4.3 GGET Statement 291
5.4.4 GPUT Statement 293
5.4.5 CONSOLE Statement 295
5.4.6 PMODE Statement 298
5.4.7 PINPUT Statement 300
5.4.8 HCOPY Statement 301
5.4.9 POINT Function 302
5.5 Application Examples 303
5.5.1 Line Chart 303
5.5.2 Bar Chart 305
5.5.3 Pie Chart 306

6. **Error Messages** 310

Appendix 1 Character Codes 316

Appendix 2 Reserved Words 317

Appendix 3 Commands 318

Appendix 4 Syntax Table 319

Appendix 5 Display Control Codes 322

Appendix 6 Calling a Machine Language Program 326

1. Program Elements

This section gives definitions, limitations, and detailed explanations of the elements used in preparing Canon BASIC language programs.

1.1 Program Lines

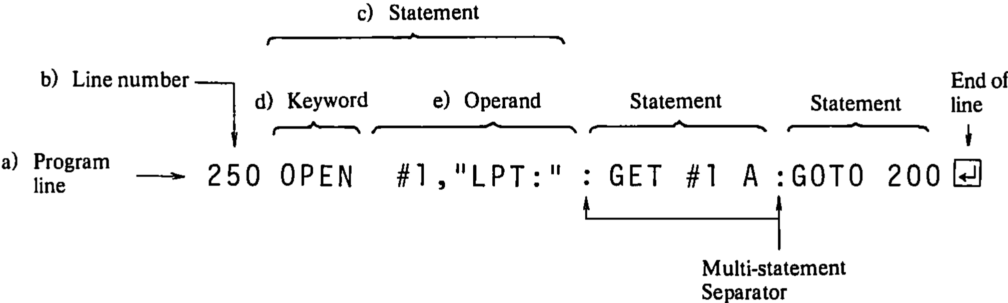
As shown below, each program consists of program lines with numbers.

```
10 DIM A(100)
20 OPEN #1,"A:DFILE.DAT"
30 REM ++DATA READ++
40 I=I+1
60 GET #1 A(I):GOTO 40
70 FOR J=1 TO I-1
   ~
```

Program lines are executed in numerical order unless there are branch directives.

A program line can contain several statements. Statements on the same line are executed from the beginning of the line. The statements are separated by a colon (:). A line consisting of more than one statement is called a multi-statement line. Depress ⏎ at the end of each line.

A statement is the unit by which instructions are executed. Each statement consists of a keyword and an operand. A keyword is a word consisting of capital alphabet letters that indicates the instruction's function. An operand is the part in which the detailed contents of the instruction are specified. These elements are described in detail below.



a) Program line . . . . A program line can contain up to 127 characters.

b) Line number . . . There must be a line number at the beginning of each line. The number must be an integer within the range: 1~32767. During program creation/editing, line numbers are assigned automatically by the automatic numbering function of the editor.

c) Statement . . . . . . Any number of statements can be included on a program line as long as the total number of characters does not exceed 127. Some types of statements do not allow multi-statements. A colon (:) mut be placed between statements on the same line.

d) Keyword . . . . The keyword is determined according to the function of the instruction and is defined in Canon BASIC.

e) Operand . . . . The operand is specified according to the syntax rules for each statement.

1.2 Constants

Data used in programs are divided into the following three types:

1) Real Number-Type Constant

This is a number within the range: $1 \times 10^{-64} \leqq x < 1 \times 10^{64}$. It can be assigned to a real number-type variable as described later. It is treated as 8-byte data.

Real number-type constants can be specified with either of the following two types of notation.

Floating type . . . . Indicated with a real number of up to 14 digits.

Examples: 1.234, −0.2345, 10000000

E type. . . . . Indicated by a real number with a mantissa of up to 14 digits and an exponent of up to 2 digits ($-64 \leqq x < 64$).

Examples: 1.23E12, −5.687E−12

2) **Integer-Type Constant**

This is an integer with the range : $-32768 \leqq x \leqq 32767$. It can be assigned to an integer-type variable as described later. It is treated as 2-byte data.
Examples: 123, −5232, 1000

Note: An integer within the range $-32768 \leqq x \leqq 32767$ can be assigned to either a real number-type or an integer-type variable. It is processed according to the type of variable to which it is assigned.

3) **String Constant**

This is character string consisting of 1-byte characters. It can be assigned to a string variable as described later. A string constant must be enclosed by double quotation marks (").
Examples: "ABC", "1234" (different than the numerical value 1234)

A string constant can also be indicated by a hexadecimal figure in ASCII code. In this case, prefix the symbol "&" to the 2-digit hexadecimal code. Refer to "Appendix 1. Character Codes" for the ASCII Codes.
In this manual, 2-digit hexadecimal code (1 byte) are indicated in the format "XX_H".

Examples: "&41" → "A", "&31&32&33" → "123"

Note: Specify "" or && respectively when a quotation mark (") or "&" is used as a character in a string constant.

Examples: "A""12""+" "A&&B"

⇓ ⇓

A"12"+ A&B

1.3 Variables

A variable is used to temporarily store data in a program for processing.

The name of a variable is specified by a character string of up to 32 alphabet letters and numbers, beginning with an alphabet letter. Keywords, sub-keywords, etc., called reserved words, cannot be used for variable names. (See "Appendix 2. Reserved Words".)

Variables are divided into the following three types:

1) Real Number-Type Variable

This is an 8-byte variable that can store a real number-type arithmetic value.

Examples: ABC, DAT1

2) Integer-Type Variable

This is a 2-byte variable that can store an integer-type arithmetic value. The name of the variable must be defined in advance by the INTEGER statement. (See "2.2.4 INTEGER Statement.")

Examples: ABC, DAT1 after execution of "INTEGER ABC, DAT1"

3) String Variable

This is a variable (usually 8 bytes) that can store a character string. In a string variable, 1 byte has a 1 character capacity. For example, an 8-byte string variable can store a string of 8 characters. The length of a string variable can be defined within the range 1~255 bytes by the DIM statement. (See "2.2.2 DIM statement".)

Suffix the symbol "$" to the string variable name.

Examples: ABC$, NAME1$

All real number-type and integer-type variables are arithmetic variables.

The same name cannot be used for variables even if their types are different.

Correct variable names: A, XYZ, VERTICAL, NAME$

Incorrect variable names:
- 12XY...... A number cannot be used as the first character.
- LET....... Reserved words cannot be used.
- $ABC ..... "$" must be suffixed to the variable name.

1.4 Array Variables

Array variables are convenient when handling a group of the same kind of data items, because all data items in the group can be handled using the same variable name. All array variables with the same name can be processed at once in the PUT statement, the GET statement, and matrix instructions.

An array variable is defined using the DIM statement.

Example: `10 DIM ABC(40)`

Executing the example above defines an array variable with 40 elements, from ABC(1)~ABC(40). It is possible to alter the starting subscript to 0 using the OPTION BASE statement. Refer to the explanations of the DIM statement and the OPTION BASE statement for details.

Real number-type variables, integer-type variables, and string variables can all be defined as array variables. Array subscripts must be integers in the range 0~32767.

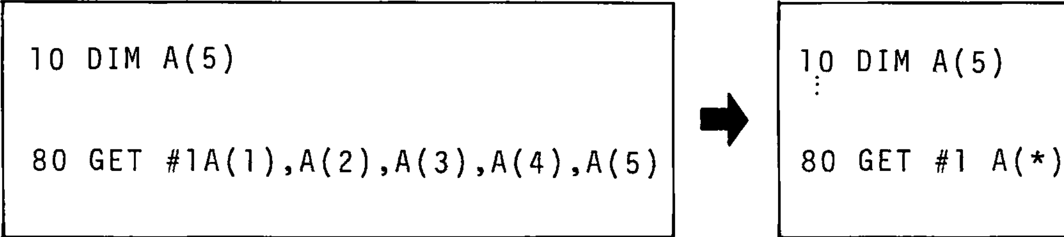
In array subscripts there is no restriction on dimension. It is possible to use any dimension of array subscripts within the range allowed by the memory capacity.

Defining array variables:

DIM A(3) → A(1), A(2), A(3)
DIM X(2,2) → X(1,1), X(1,2), X(2,1), X(2,2)

↑
Two-dimension array

An asterisk (*) can be specified instead of a subscript to handle all defined array variables as one block. When an asterisk is specified, all array variables with the same name are handled as a block regardless of their dimension.



For example, array variables of more than one dimension can be specified with an asterisk that corresponds as follows:

Under the definition of DIM X(3,3):
X(*) ⟹ X(1,1), X(1,2), X(1,3), X(2,1), X(2,2), X(2,3), X(3,1), X(3,2), X(3,3)

General specification of array variables using an asterisk is available for the following statement:

CALL, PARAM, MAT, PUT, GET.

1.5 Arithmetic Operators

Arithmetic operations in a program are performed using the operators listed below. An operation which mixes data of different types causes an error. (Integer-type and the real number-type variables can be mixed.)

O Addition	+
O Subtraction	−
O Multiplication	*
O Division	/
O Power	** (When X<0, Y must be an integer in power calculations X**Y.)
O Priority operation	(,)

The calculation is carried out according to the following priority: parentheses, power, multiplication or division, and addition or subtraction. The addition operator also connects character strings.

"ABC"+"XYZ" → "ABCXYZ"

1.6 Relational Operators and Relational Expressions

Relational operators that are used to compare the values of data in programs are listed in the following table.

Relational operator	Representation
= (Equals)	=
> (Greater than)	>
≧ (Greater than or equal to)	>=
< (Less than)	<
≦ (Less than or equal to)	<=
≠ (Not equal to)	< >

Relational operators compare either two numeric values and or two characters. A numeric value and a character string cannot be compared. Characters are compared based on their values in ASCII code.

An expression using relational operators is called a relational expression. In relational expressions the value of the expression is −1 (true) when the conditions are satisfied. When the conditions are not satisfied, the value of the expression is 0 (false). Program examples using relational expressions are shown below. Refer to the explanations of the IF and LET statements for details.

Relational expressions and logical expressions which have specified conditions, described later, are also called conditional expressions.

[Ex. 1.6-1]

```
90 IF A>0 GOTO 300
```

When variable A on line 90 is a positive value, the condition of the IF statement is satisfied and program execution branches to line 300.

[Ex. 1.6-2]

```
100  X=-((A>0)+(B>0)+(C>0))
```

A check is performed to determine the values of A, B, and C.
If A=3, B=−1, and C=−5
Then A>0: true (−1), B>0: false (0) and C>0: false (0).
So −((−1)+0+0) is calculated and 1 is assigned to X.

1.7 Logical Operators and Logical Expressions

Logical operators are used to specify logical expressions (compare various expressions) used in IF statements, etc. The following four types of logical operators are available:

Logical operator	Format
AND (Logical product)	<Expression 1>AND<Expression 2>
OR (Logical sum)	<Expression 1>OR<Expression 2>
XOR (Exclusive OR)	<Expression 1>XOR<Expression 2>
NOT (Negation)	NOT <Expression 1>

1) AND (Logical Product)

The result is true (−1) only when the conditions of <Expression 1> and <Expression 2> are both true. Otherwise the result is false (0).

When <Expression 1> and <Expression 2> are both arithmetic expressions (described later), their values are first converted to 2-byte integer type values. Then they are compared bit by bit. The resulting bit is 1 only when the two corresponding bits are 1. Otherwise the resulting bit is 0.

2) **OR (Logical Sum)**

The result is false (0) only when the conditions of <Expression 1> and <Expression 2> are both false (0). Otherwise the result is true (—1).

When both <Expression 1> and <Expression 2> are arithmetic expressions, their values are first converted to 2-byte integer type values. Then they are compared bit by bit. The resulting bit is 0 only when the corresponding bits are 0. Otherwise the resulting bit is 1.

3) **XOR (Exclusive OR)**

The result is false (0) only when the conditions of <Expression 1> and <Expression 2> are both true (—1) or both false (0). Otherwise the result is true (—1).

When <Expression 1> and <Expression 2> are both arithmetic expressions, their values are first converted to 2-byte integer type values. Then they are compared bit by bit. The resulting bit is 0 only when the corresponding bits agree. Otherwise the resulting bit is 1.

4) **NOT (Negation)**

The result is false (0) when the condition of <Expression 1> is true (—1). The result is true (—1) when the condition is false (0).

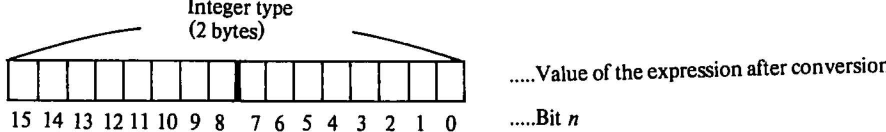
When <Expression 1> is an arithmetic expression, it's value is first converted to a 2-byte integer type value. Then the value of each bit of the 2-byte value is inverted. The resulting bit is 0 when the bit is 1. The resulting bit is 1 when the bit is 0.

1. When expressions are conditoinal:

Expression 1	Expression 2	AND	OR	XOR
True (—1)	True (—1)	True (—1)	True (—1)	False (0)
True (—1)	False (0)	False (0)	True (—1)	True (—1)
False (0)	True (—1)	False (0)	True (—1)	True (—1)
False (0)	False (0)	False (0)	False (0)	False (0)

Expression 1	NOT
True (—1)	False (0)
False (0)	True (—1)

2. When expressions are arithmetic:



Bit n after conversion of expression 1	Bit n after conversion of expression 2	Bit n in result of AND	Bit n in result of OR	Bit n in result of XOR
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Bit *n* after conversion of expression 1	Bit *n* in result of NOT
1	0
0	1

Expressions specified using AND, OR, XOR, and NOT are called logical expressions. Program examples using logical expressions are given next. Refer to the explanations of the IF and LET statements for details.

In all of the examples, the value of variable A is 5 and the value of variable B is 3.

[Ex. 1.7-1]
Conditional expression of AND

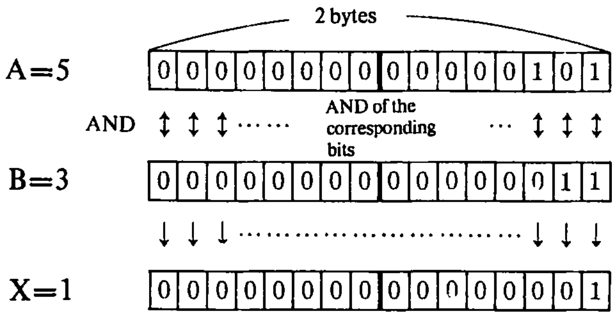
```
90 IF A>0 AND B=3 GOTO 50
```

On line 90, A>0: true (5>0) and B=3: true (3=3), so the result of the expression is true. The condition of the IF statement is satisfied and program execution branches to line 50.

[Ex. 1.7-2]
Arithmetic expression of AND

```
90 LET X=A AND B
```

On line 90, bits are compared as shown below and 1 is assigned to variable X.



[Ex. 1.7-3]
Conditional expression of OR

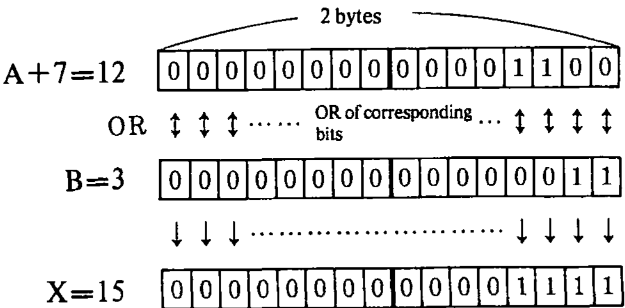
```
100 IF A<=0 OR B=3 GOTO 50
```

On line 100, A≦0: false (5>0) and B=3: true (3=3), so the result of the expression is true. The condition of the IF statement is satisfied and program execution branches to line 50.

[Ex. 1.7-4]
Arithmetic expression of OR

```
110 LET X=A+7 OR B
```

On line 110, bits are compared as shown below and 15 is assigned to variable X.



[EX. 1.7-5]
Conditional expression of XOR

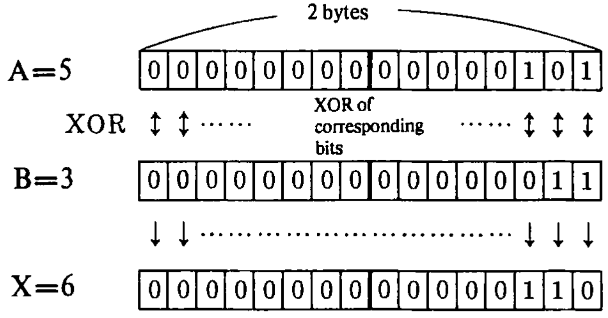
```
60 IF A-5=0 XOR B-3>=0 GOTO 50
70 ..
```

On line 60, A−5=0: true (5−5=0) and B−3≧0: true (3−3=0), so the result of the expression is false. The condition of the IF statement is not satisfied and program execution proceeds to the next line.

[Ex. 1.7-6]
Arithmetic expression of XOR

```
70 X=A XOR B
```

On line 70, bits are compared as shown below and 6 is assigned to variable X.



[Ex. 1.7-7]
Conditional expression of NOT

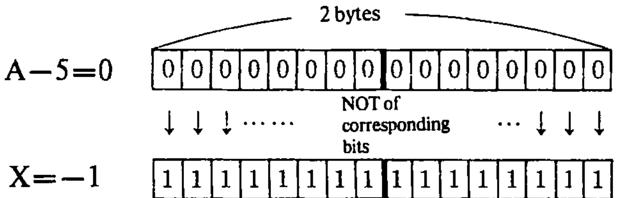
```
120 IF NOT A>0 GOTO 50
130........
```

On line 120, A>0: true (5>0), so the result is the negation of this, false. The condition of the IF statement is not satisfied and program execution proceeds to the next line.

[Ex. 1.7-8]
Arithmetic expression of NOT

```
40 X=NOT A-5
```

On line 40, bits are inverted as shown below and −1 is assigned to variable X.



*A negative value of integer-type data is expressed as 2.5 complement.

1.8 Arithmetic Expressions

Arithmetic expressions are expressions that represent values (real number-type and integer-type constants), arithmetic variables, and those values and variables combined by arithmetic, relational, and logical operators, etc. Logical and relational expressions are considered as arithmetic expressions because they have a value of −1 when their result is true and 0 when their result is false.

The types of arithmetic expressions are shown below.

324 .... Integer-type constant
1.5E23 .... Real number-type constant
A............... Arithmetic variable
A+45
A*(B+C)
A AND B......... Logical expressions
C>=45.......... Relational expression

Calculation results of arithmetic expressions vary depending on the types of constants and variables (integer-type and real number-type) and also upon arithmetic operations.

1) Addition, subtraction and multiplication:
Integer-type and integer-type....
The result is an integer-type value. When the result exceeds the integer-type range ($-32768 \leqq x < 32767$), it is automatically converted to a real number-type value.

Integer-type and real number-type or real number-type and real number-type. . . .
The result is a real number-type value. Integer-type values are automatically converted to real number-type values before calculation.

2) Division and power:
All results are real number-type values. Integer-type values are automatically converted to real number-type values before calculation.

1.9 String Expressions

String expressions consist of string constants, string variables, and character strings and combine those constants and variables.

Various string expressions are shown below.

"ABC" . . . . String constant
"&41&42"
NAME$. . . . . String variable
A$+B$
C$+"XYZ"

1.10 Files

Files are classified into two general categories—program files and data files. There are also files managed exclusively by the operating system.

Each file has a name. File names must be specified when handling the file. File names can be expressed with up to 8 capital alphabet letters or numbers, beginning with a capital letter.

Up to 128 files can be stored on a disk.

Refer to the "CP/M-86 User's Manual" for details about disk handling.

Data is read from or written to a data file in a program by specifying a logical device number (described next). At this time, the operating system reads/writes data from/to files in 128-byte units. For details about processing, refer to "2.12 File-Related Instructions."

1.11 Logical Device Numbers

Numbers must be defined in advance to specify execution of input/output from/to files on disks or external peripheral devices. These numbers are called logical device numbers. Logical device numbers are defined by the OPEN statement which is described later. The definitions are canceled by the CLOSE statement.

Nine logical device numbers (1~9) can be defined. So up to 9 data files and/or I/O devices can be defined at the same time.

[Ex. 1.11-1]

```
40  OPEN #1,"A:DAT1"
      :
140 PUT #1 A,B,C
      :
400 CLOSE #1
```

The OPEN statement on line 40 in the above program defines the data file "DAT1" in drive A as logical device number 1. On line 140, the PUT statement is executed against logical device number 1 and the contents of the variables A, B, and C are written to the data file. The CLOSE statement on line 400 cancels the definition.

Refer to "2. Instructions" for details of the respective statements.

2. Instructions

This section explains the functions, formats, procedures, etc. of program instructions using program examples.

Each explanation contains the following:

- •Heading: The keyword and the full name of the instruction are shown.
- •Function: The function of the instruction is described briefly.
- •Format: The syntax of the instruction is shown. For the rule of interpretation. Refer to "2.1 Formats".
- •Explanation: The details of the function, procedures for its use, limitations, etc. of the instruction are explained. The most important parts are underlined.
- •Note: The points that could lead to the misuse of the instruction are emphasized.
- •Advice: Programming techniques and other hints are given for more efficient use of the instruction.
- •Example: A program example is given to explain how to use the instruction. Lines not needed for explanation are omitted. Line numbers are for the sake of convenience only.

2.1 Formats

This section explains individual program instructions. Prior to the explanation, the rules for interpreting the formats of program lines is described.

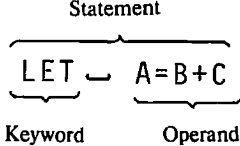
2.1.1 Program Instruction Elements

Instruction word
or keyword .... Indicates the function of the instruction. It consists of capital alphabet letters.

Operand .... Specifies the detailed contents of the instruction. It is written after the keyword.

Statement........... An instruction consisting of a keyword and the operand that specifies an action.

[Example]



The example above is the LET statement consisting of the keyword LET and the operand A=B+C.

2.1.2 Symbols Used in Formats

< > ..... Indicates one element in an operand.

[ ] ...... Indicates that the enclosed element can be omitted.

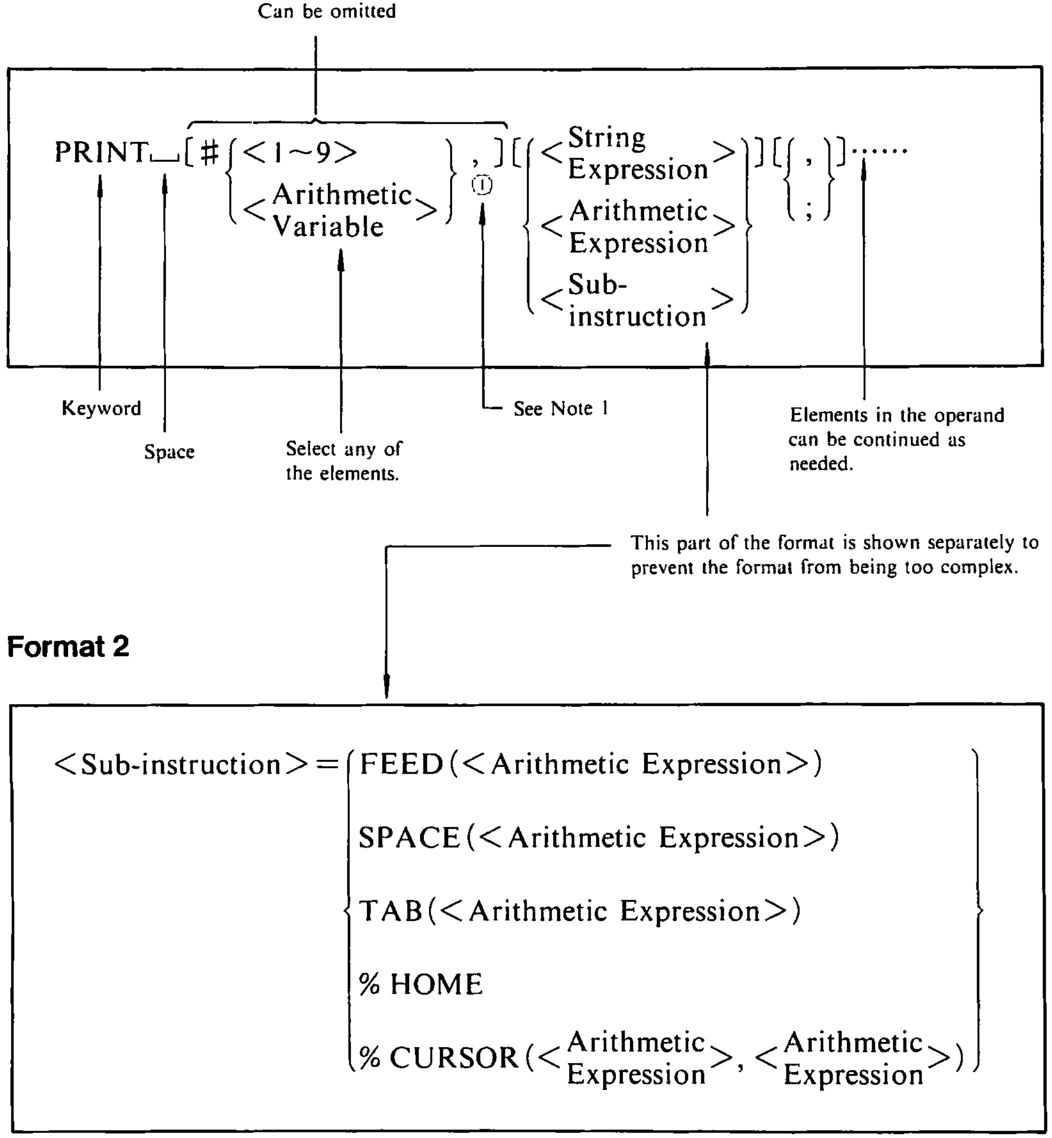
{ } ...... Any one of the elements between the brackets can be selected.

.... Indicates that the operand can be repeated as necessary.

␣ ...... This indicates one or more spaces.

2.1.3 Format Interpretation Examples

Format 1



Note 1: The comma at the end of a statement may be omitted.

2.2 Declaration Instructions

2.2.1 REM Statement (Remark)

Function

The REM statement inserts a comment into a program list. This statement does not affect program execution.

Format

```
REM ␣ <Comment>
```

Explanation

This is a statement that the programmer uses to insert comments into the program to clarify the program list.

Because this instruction is stored as part of the program, it uses memory area in proportion to the length of the comment.

It is indicated as a program line during program editing and listing, but it does not affect program execution.

Any character that can be entered through the keyboard can be used in the comment. The REM statement can include up to 127 characters, including the line number, keyword, and ↵.

There must be at least one space between the keyword and the comment.

[Ex. 2.2.1-1]

The program title is entered at the head of the program.

```
10  REM TEST PROGRAM NO.1
```

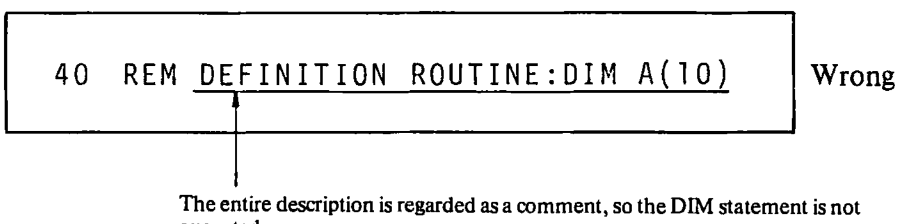
[Ex. 2..2.1—2]

A comment is inserted into each routine to describe the sections of a program.

```
10  REM PROGRAM START
20  REM INPUT ROUTINE
 :
150 REM CALCULATION ROUTINE
 :
300 REM OUTPUT ROUTINE
```

Note All characters in the statement following the keyword REM are regarded as part of the comment. This means that any statements on the same line following REM are not executed.

[Example]



```
40  REM DEFINITION ROUTINE:DIM A(10)
```

Wrong

The entire description is regarded as a comment, so the DIM statement is not executed.

```
40  DIM A(10):REM DEFINITION ROUTINE
```

Right

2.2.2 DIM Statement (Dimension)

Function

The DIM statement defines array variables and string variables of an irregular length and reserves space in memory for them.

Format

```
DIM ␣ <Variable> [ , ]······
```

Explanation

When array variables are used in a program or when string variables of a length of other than 8 bytes are used, this statement must be executed to define the uses of the variables and reserve memory areas for them. Simple variables (variables without subscripts) and 8-byte string variables can be used without definition by the DIM statement.

A variable can only be defined once by this statement. More than one definition causes an error.

Definition of Array Variables

Array variables are defined by specifying the variable name with the highest subscript value in the operand. For example, specifying A(10) defines the 10 array variables, A(1)~A(10).

Because the highest subscript value allowed for array variables is 32767, specifying a higher value causes an error. The array dimension (1-dimension: A(10); 2-dimension: A(10, 10) . . .) is unlimited. An error occurs if there is insufficient memory area for the variable that will be defined by this statement.

Definition of String Variables

String variables have an initial length of 8 bytes. But it is possible to change the length to accommodate strings of irregular lengths, for example, 3 characters or 10 characters.

Not only does this save memory area but sometimes it even makes processing easier.

To define the length of an irregular string variable, specify the necessary number of bytes immediately following the variable name. For example, specifying A$30 in the DIM statement redefines the length of string variable A$ as 30 bytes. Only the variable name must be used in the program after definition by the DIM statement. The length need not be specified.

String variables can be defined within the range: 1—255 bytes. An error occurs if a length outside this range is specified.

Advice

As you now know, array variables and string variables of irregular lengths cannot be used unless they are defined by the DIM statement. The DIM statement can be executed anytime before such variables are used in the program. It is best, however to define all variables at the beginning of the program to prevent duplicate definitions, to reserve necessary memory area in advance, and to clarify the types of variables that will be used in the program.

[Ex. 2.2.2-1]

Array variables A(1,1)~A(2,2), B(1)~B(4), and C$(1)~C$(4) are defined.

```
10 DIM A(2,2),B(4),C$(4)
```

In this example, the following 12 array variables are defined:

A(1,1), A(1,2), A(2,1), A(2,2)
B(1), B(2), B(3), B(4)
C$(1), C$(2), C$(3), C$(4)

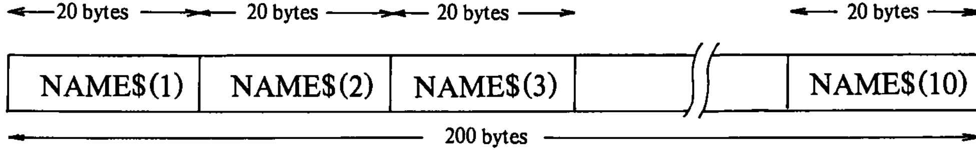
Total: 12 variables

[Ex. 2.2.2-2]

String array variables NAME$(1)~NAME$(10) are defined as 20 bytes (20 characters) per variable.

```
10 DIM NAME$20(10)
```

Memory is reserved as shown below.



[Ex. 2.2.2-3]

Three-dimensional array variables M(1,1,1)~M(2,2,2) are defined.

```
10 DIM M(2,2,2)
```

The following 8 variables are defined in the statement above.
M(1,1,1), M(1,1,2), M(1,2,1)~M(2,2,2)... 8 variables

[Ex. 2.2.2-4]

The number of array variables is specified by value of a variable.

```
10 INPUT N
20 DIM A(N)
```

In this example, the value input for variable N on line 10 specifies the range of array variables that will be defined by the DIM statement on line 20. (The INPUT statement is described later.)

For example, if "10" is entered through the keyboard during execution of the INPUT statement on line 10, 10 array variables, A(1)~A(10), are defined by the DIM statement on line 20. If a decimal fraction (e.g. 5.7) is entered for N, the DIM statement defines array variables by automatically truncating the fractional part of the value of N.

2.2.3 OPTION BASE Statement (Option Base)

Function

The OPTION BASE statement specifies the subscript for the first element of an array as 0.

Format

```
OPTION␣BASE␣0
```

Explanation

The subscript of an array variable is initially set to 1. This statement changes the starting subscript to 0. For example, the statement "DIM A(2)" usually defines two variables, A(1) and A(2), but three variables, A(0) through A(2), are defined after executing the OPTION BASE statement. This instruction must be executed before the DIM statement. Execution of this instruction more than once in a program causes an error.

[Ex. 2.2.3-1]
Specify that the subscript of an array variable start with 0.

```
10 OPTION BASE 0
20 DIM A(3),B(2,2)
```

The following array variables are defined in the above example.

A(0), A(1), A(2), A(3) . . . 4 variables

B(0,0), B(0,1), B(0,2)
B(1,0), B(1,1), B(1,2)
B(2,0), B(2,1), B(2,2) . . . 9 variables
Total: 13 variables

Without the OPTION BASE statement on line 10, 7 array variables, A(1), A(2), A(3), B(1,1), B(1,2), B(2,1), and B(2,2) are defined.

Note

If this statement is executed somewhere in a program, the subscript of any array variable defined prior to the execution also starts with 0. For example, an array A(1)~A(3) is A(0)~A(2) after the execution of this instruction.

2.2.4 INTEGER Statement (Integer)

Function

The INTEGER statement defines the name of an integer-type variable.

Format:

```
INTEGER␣<Arithmetic Variable>[,]……
```

Explanation

This instruction defines the name of an integer-type variable. The variable whose name is specified in the operand of this statement is treated as an integer-type variable after this statement is executed.

An integer-type variable has a length of 2 bytes. Values assigned must be integers within the range: $-32768 \leqq x \leqq 32767$. Assigning a value outside this range causes an error. When a decimal fraction within this range (e.g. 7.25) is assigned, the fractional part is truncated automatically. (That is, 7 is assigned.)

To define an array variable as an integer type-variable, execute an INTEGER statement that specifies only the name of the array variable (without the subscript) before executing the DIM statement that defines the variable.

Advice

When a variable is defined as an integer-type variable, the range of values that can be assigned is limited. But it does save memory and increase the processing speed. Check the values that will be input to variables carefully during program design. Those values which can be processed within the value range of integer-type variables (e.g. employee numbers, etc.) should be processed as integer-type variables.

[Ex. 2.2.4-1]
Integer-type variables R, S, and T are defined.

```
10 INTEGER R,S,T
```

[Ex. 2.2.4-2]
Ten integer-type array variables AREA are defined.

```
10 INTEGER AREA
20 DIM AREA(10)
```

In this example, 10 integer-type array variables, AREA(1)~AREA(10), are defined.

Note

Reversing the order of line 10 and line 20 in above example causes an error. This is because if the DIM statement is executed first, the real number-type array variables having a length of 8 bytes are defined then and memory is reserved for that length. Even if the INTEGER statement is executed later, the real number-type variable already defined cannot be converted to an integer-type, so an error occurs.

[Ex. 2.2.4-3]
The fractional part of the data is truncated when an integer-type variable is used.

```
10 INTEGER UNIT
20 DIM UNIT(10)
        ⋮
80 LET UNIT(1)=X
```

If data with a decimal fraction is assigned to an integer-type variable, the fractional part is truncated automatically. On line 80 of this example, the value of real number-type variable X is assigned to integer-type variable UNIT(1) by the LET statement (described later). When the value of X is 14.25, 14 is assigned to UNIT(1).

2.2.5 DEFKEY Statement (Define Key)

Function

The DEFKEY statement defines a character string entered when a function key is depressed in the shift down mode.

Format

```
DEFKEY ␣ <Arithmetic Expression>, <String Expression>
```

Explanation

In the initial state, various command names are defined for both shift up and shift down mode of the 12 function keys (F1 ~ F12). The DEFKEY statement changes the definitions of the function keys in the shift down mode (in which the shift key is not depressed) and permits input of any character string using the function key operation.

The number of the function key (1 ~ 12) is specified in the arithmetic expression part of the operand. In the string expression part, the character string that will be defined for the function key is specified. The character string cannot exceed 15 characters. A control code may be included. Only the first 15 characters specified are valid and excess characters are ignored.

The character string defined is valid until the definition is changed using this statement or BASIC is ended (i.e. control is returned to the operating system or the power is turned off).

If a decimal fraction within the range: $1 \leqq x < 13$ is specified as a function key number, the fractional part is automatically truncated. Specifying a value outside this range causes an error.

[Ex. 2.2.5-1]

The character string "CANON" is defined for function key 12.

```
:
50 DEFKEY 12,"CANON"
:
```

In this example, the character string "CANON" is defined for [F12]. When [F12] is depressed in the shift down mode (the shift key is not depressed) after execution of the DEFKEY statement on line 50, the input is exactly the same as if [C] [A] [N] [O] [N] is depressed.

[Ex. 2.2.5-2]

A function key definition program is prepared to permit the selection of a program that will be executed by function keys and the [↵] key.

```
10 DEFKEY 1,"PROG1"
20 DEFKEY 2,"PROG2"
30 DEFKEY 3,"PROG3"
40 END
```

After this program is executed, depressing [F1], [F2], and [F3] in the shift down mode produces the same result as the inputting the program name. [F1] corresponds to the operation for inputting program name "PROG1", [F2] to that of "PROG2", and [F3] to that of "PROG3". So each program is executed by depressing one of the three function keys and [↵]. This definition remains valid until control is returned to the operating system or the power is turned off.

Advice

Use the function key overlays on the keyboard to write the definitions so that programs and jobs can be selected easily.

2.3 Assignment Instruction

2.3.1 LET Statement (Let)

Function

The LET statement assigns data to a variable.

Format

```
[LET ␣]<Variable>= { <Arithmetic Expression> }
                   { <String Expression>     }
```

Explanation

The value on the right side of the operand is assigned to the variable on the left side. The equal sign (=) in the operand of the LET statement means to assign the value on the right side to the variable on the left side.

The type of data on the right side of the operand must be the same as that on the left. So, if the left side is a string-type variable, the right side must be a string expression. If the type of data on the right and on the left side do not agree, an error occurs.

The keyword LET can be omitted. Any operand consisting only of the right side and the left side with an equal sign in between is treated as a LET statement.

Some built-in functions can be specified as a variable on the left side. Refer to "3. Built-in Functions".

Note

Using an array variable not defined in advance by the DIM statement causes an error.

[Ex. 2.3.1-1]

The value 26 is assigned to arithmetic variable A.

```
30 LET A=26
```

When line 30 is executed, 26 is assigned to variable A. The same result is obtained when the example below, in which the keyword is omitted, is executed.

```
30 A=26
```

[Ex. 2.3.1-2]

The right side is calculated using the value of variable W and the result is assigned to variable A.

```
40 LET A=W*3.14+440
```

If the value of W is 100, the value of A is 754.

[Ex. 2.3.1-3]

The characters "JOHN" are assigned to string variable NAME$.

```
90  LET NAME$="JOHN"
```

Initially up to 8 characters can be assigned to a string variable. In this example, 4 characters are assigned to the string variable (8-byte). When as in this case, fewer than 8 characters are assigned, the NUL code (00_H) is automatically added to "JOHN".

When more than 8 characters are assigned, only the first 8 characters entered are assigned and excess characters are ignored.

Note The NUL code (00_H) is not treated as data.

[Ex. 2.3.1-4]

The result of a calculation using variable M is again assigned to variable M.

```
60 LET M=M*12
```

[Ex. 2.3.1-5]

The character data in string variables B$ and C$ are connected to assign them to string variable A$.

Example of data:

"SIZE·L"="SIZE"+"·L"

```
70 LET A$=B$+C$
```

[Ex. 2.3.1-6]

The values of variables A, B, and C are compared using a relational operator, and the result of the conditional expression is assigned to variable D.

```
90 LET D=-((A>B)+(A>C))
```

In this example, the following values are assigned to variable D, according to the values of the variables A, B, and C.

$A>B$, $A>C$ 2
$A\leqq B$, $A>C$ 1
$A>B$, $A\leqq C$ 1
$A\leqq B$, $A\leqq C$ 0

When the value of the conditional expression is true, the value is −1, and when the expression is false, the value is 0.

2.4 Input Instructions

2.4.1 INPUT Statement (Input)

Function

The INPUT statement reads data entered through the keyboard and assigns it to a variable. Data from a disk file or an external input device can be assigned to a variable by specifying the logical device number.

Format

```
INPUT␣[#{<1~9>              }, ][MSG(<String Expression>)][<Variable>][,]......
        {<Arithmetic Variable>} ①
```

Note 1: The comma (,) at the end of the statement can be omitted.

Expalanation

Any characters, numbers, or symbols entered through the keyboard can be assigned to variables using the INPUT statement. The ↵ key must be depressed at the end of input. Depressing this key indicates that input to a variable is completed.

Data separated by commas can be assigned to two or more specified variables at the same time.

When this statement is executed without logical device number specification, "?" is displayed. An input operation through keyboard for the INPUT statement at that time is echoed back following "?". When ↵ key is depressed, the data is assigned to the variable. Data can be corrected before ↵ is depressed with DEL, DELETE LINE, DELETE, or INSERT. Any message can be displayed instead of "?" using the sub-keyword "MSG".

If incorrect data (unmatched with the variable) is entered, the data is not assigned and entered data is displayed together with "??" to request reinput.

The INPUT statement is the main statement to control input. It has such functions as branch operation and temporary program suspension. Details of these functions are described later.

[Ex. 2.4.2-1]

Data are input to variables "LENGTH1" and "LENGTH2" through the keyboard.

```
50 INPUT LENGTH1,LENGTH2
```

In this example, keyboard input operations are as follows:

«Display»	«Operation»
?_	
	[1][2][3]
?123_	
	[↵]
?123?_	
	[4][5][6]
?123?456_	
	[↵]
?123?456	

"123" is assigned to variable LENGTH1 and "456" to LENGTH2.

These two data that will be assigned to each variable can be input at the same time using a comma as shown below. This type of input is called batch input.

«Display»	«Operation»
?_	
	[1][2][3][,][4][5][6]
?123,456_	
	[↵]
?123,456	

As shown in this example, data separated by a comma(s) can be input at the same time to an arithmetic variable specified in the INPUT statement. Batch input cannot be performed to a string variable because a comma (,) is regarded as data.

Note During batch input, when the number of data connected by commas is less than the number of the variables specified in the INPUT statement, "?" is displayed continuously, prompting input for the remaining variables. If there are more data than variables, only the data corresponding to the variables are assigned sequentially, and the excess data are ignored.

[Ex. 2.4.1-2]

Data are input through the keyboard to arithmetic variable A and string variable NAME$.

```
60 INPUT A,NAME$
```

«Display»	«Operation»
?_	
	[1][2][3]
?123_	
	[↵]
?123?_	
	[C][A][N][O][N]
?123?CANON_	
	[↵]
?123?CANON	

In this program example, 123 is assigned to arithmetic variable A and "CANON" is assigned to string variable NAME$.

Because string variable NAME$ has a length of 8 bytes, 3 NUL codes (00_H)[1] are automatically added following the data "CANON".

Only the first 8 characters entered are assigned and excess characters are ignored.

Note 1: "XX_H" is an ASCII code consisting of 2 hexadecimal figures.

Retaining Data

When ↵ is depressed without data input during execution of the INPUT statement, data is not assigned to the variable and INPUT statement execution is ended. The contents of the variable are retained. This operation is called a no-input operation. Depressing ↵ without data input changes the flow of program execution. This is called a branch operation and is explained later.

[Ex. 2.4.1-3]

```
80 LET C=328
90 INPUT C
```

During the input operation to variable C by the execution of line 90, when ↵ key is depressed without data input, 328 assigned on line 80 is retained and program execution proceeds to the next line.

To retain the data of one or more variables in batch input, input only commas as shown below.

[Ex. 2.4.1-4]

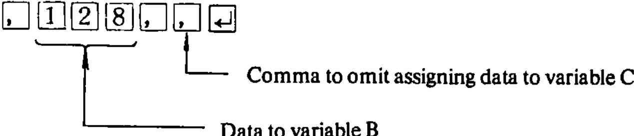
```
80 A=10:B=20:C=30
90 INPUT A,B,C
```

«Display»	«Operation»
?_	
	[,] [,] [9] [5]
?,,95_	
	↵
?,,95	

This is an example in which 95 is assigned to variable C and the contents of variables A and B are retained. When only a comma(s) is entered in batch input with data omitted, the contents of the variables corresponding to the data omitted are retained.

Note During batch input, to omit the data corresponding to the last variable, e.g. the third variable in an INPUT statement specifying 3 variables, a comma must be entered instead of data.

[Example] To input data by executing "INPUT A, B, C", enter the following to input data to variable B only.

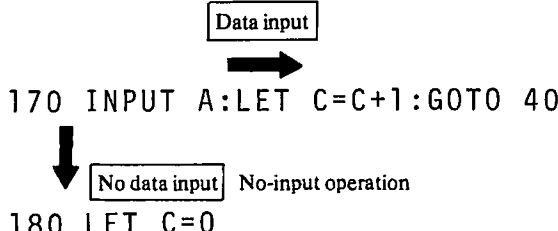


Branch Operation

The no-input operation (depressing ↵ without data input) changes the flow of program execution following an INPUT statement.

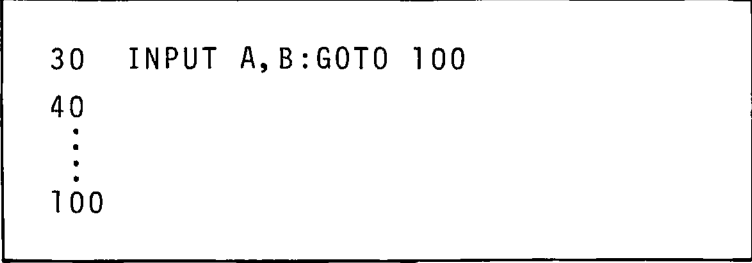
And when ↵ is depressed without data input to a variable specified in the operand of the INPUT statement, program execution proceeds to the next line. This means that the statement placed on the same line following the INPUT statement is executed only when data is entered. But when data input is omitted during batch input, the execution flow is the same as if data was entered normally.

The execution order of a branch operation is illustrated below.



[Ex. 2.4.1-5]

A program branch made by the operator's input operation.



```
30  INPUT A,B:GOTO 100
40
 :
100
```

In this program example, data are input to variable A and B using the INPUT statement on line 30. The program execution flow varies depending on whether the GOTO statement (described later) on line 30 is executed or not. The GOTO statement is a branch instruction. When the GOTO statement on line 30 is executed, program execution branches to line 100.

Assuming that the data that will be input to variable A is 10 and that the data that will be input to variable B is 20, the program is executed according to input operations as follows.

a) [1] [0] [↵] [2] [0] [↵]

In this case, 10 is assigned to variable A and 20 to variable B. Then the next GOTO statement is executed and program execution branches to line 100.

b) [1] [0] [,] [2] [0] [↵] Batch Input

(or [,] [2] [0] [↵] or [1] [0] [,] [,] [↵] or [,] [,] [↵])

Data are assigned to the variables (or retained) according to the rules for batch input. Then the next GOTO statement is executed and program execution branches to line 100.

c) [1] [0] [↵] [↵] ..... No-Input of B

In this case, 10 is assigned to variable A and program execution proceeds to the next line (line 40). The contents of variable B are retained.

d) [↵] ..... No-Input Operation

The contents of both variables are retained and program execution proceeds to line 40.

Note Even when more than one variable is specified in an INPUT statement, when no-input operation is performed, INPUT statement execution is ended and program execution proceeds immediately to the next line even if some of the variables are not assigned data.

Input with a message

When the INPUT statement is executed, it is possible to display another message instead of "?". The sub-keyword "MSG" is used to display a message.

[Ex. 2.4.1-5]
Input to variable HEIGHT1 is prompted by displaying the message "HEIGHT␣=".

```
90 INPUT MSG("HEIGHT=")HEIGHT1
```

«Display» «Operation»

HEIGHT =_ [1][5][0]

HEIGHT =150_ [↵]

HEIGHT =150

150 is assigned to variable HEIGHT1 using this operation.

[Ex. 2.4.1-6]
The value of a variable is displayed in a message.
Using the character processing function CHR$ (described later), numeric data specified as a part of the message is converted into character data.
In this example, data is input to array variable POINT(I) through the keyboard according to variable I. At this time the message "POINT␣OF␣NO=I?␣" is displayed.

```
170 INPUT MSG("POINT OF NO="+CHR$(I)+"? ")POINT(I)
```

When I=5;

«Display» «Operation»

```
POINT OF NO=_5?_
                                    [9][5]
POINT_OF_NO=_5?_95
                                    [↵]
POINT_OF_NO=_5?_95
```

As a result of this operation, 95 is assigned to variable POINT(5). The space just before "5" in the message is automatically inserted in front of the character string (number) when the numeric value is converted by the CHR$ function.

Temporary Program Suspension

Unless the name of a variable is specified in the operand of the INPUT statement, program execution is suspended only temporarily and program execution is restarted by depressing [↵] . This is called the pause function. Specify the following for no display.

```
INPUT MSG("")
```

A branch operation is valid even if the variable is not specified in the INPUT statement. When only [↵] is depressed, program execution proceeds to the next line. When data is entered before depressing [↵], the input data are ignored and program execution proceeds to the following statement on the same line.

Input from Disk Files

Data can be read from a disk file or an external input device and assigned to a variable by specifying a logical device number in the INPUT statement. For details, refer to the explanation of the INPUT statement in "2.12.6 Other Input/Output".

2.4.2 INPUT USING Statement (Input Using)

Function

The INPUT USING statement enters data to variables by specifying the number of digits or characters that will be entered.

Format

```
INPUT ␣ [ # { <1~9>                  } , ] USING ␣ { <Line No.>      } ␣ <Variable> [ , ] ……
            { <Arithmetic Variable>  }              { [ <Label> ]     }
```

Explanation

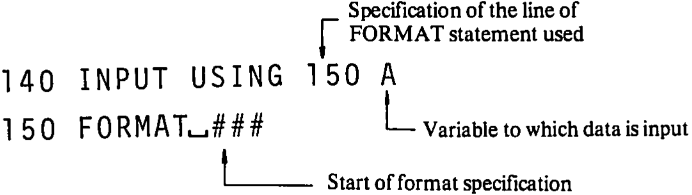
The basic function is the same as that of the INPUT statement, but the number of digits or characters that will be entered is specified in advance. Input is completed when the specified volume of data is entered. No input prompting symbol or cursor is displayed.

The input format is specified by the FORMAT statement. The line on which the FORMAT statement is written must be specified following USING in the INPUT USING statement. Executing the FORMAT statement by itself has no effect.

The FORMAT statement is also specifies the format for the PRINT USING statement.

Input Specification Using the FORMAT Statement

In the FORMAT statement used with the INPUT USING statement, the input format is specified by "#". Look at the following example using the INPUT USING statement and the FORMAT statement:

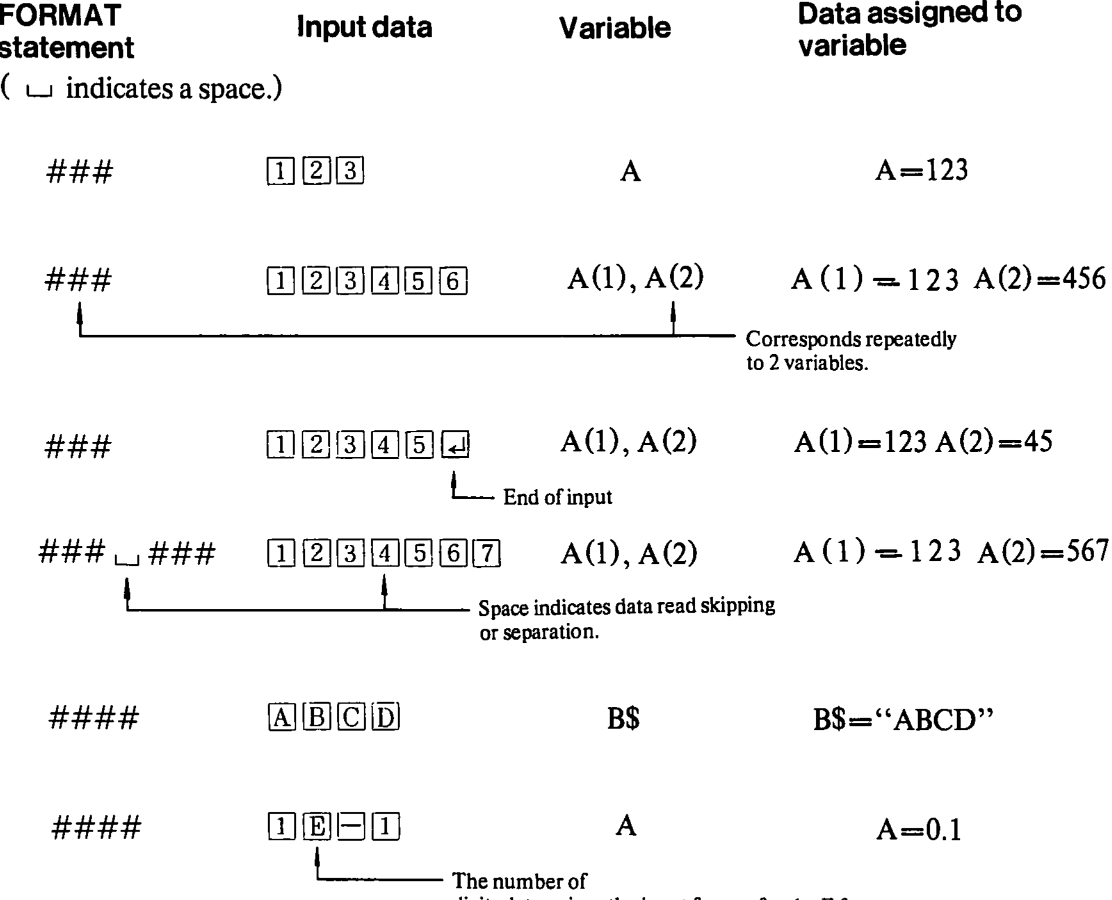


In the FORMAT statement, specification of the input format starts after a space is entered following the keyword. In the above example, the input of 3 digits is specified for variable A in the INPUT USING statement on line 140.

So when the INPUT USING statement on line 150 is executed, a 3-digit data can be input to the variable A. When [1], [2], and [3] are depressed, for example, input to variable A is completed. After assignment of 123 to variable A, program execution proceeds to the next statement.

Depressing [↵] is unnecessary when inputting data with the INPUT USING statement, because the number of digits or characters is specified by the FORMAT statement.

The relationship between the specification of the input form by the FORMAT statement and the data assigned to the variable specified in the INPUT USING statement is shown next.



Data that can be entered using the INPUT USING statement are limited as follows according to the variable type.

Arithmetic variable:
Only the number of digits input must be considered, regardless of whether the variable is of the real number-type or the integer-type. Inputting invalid data causes an error.

String variable:
The data are characters, numbers, and symbols that can be entered through the keyboard.

Note Using characters other than "#" or a space in the FORMAT statement used with the INPUT USING statement causes an error. But when the FORMAT statement is used with the PRINT USING statement, other characters are valid. So, do not use the FORMAT statement with both the INPUT USING statement and PRINT USING statement.

[Ex. 2.4.2-1]
Data entered one after another through the keyboard are assigned to several variables.

```
100 INPUT USING 110 A,B,C,D
110 FORMAT ###
```

When the 12-digit data "123456789012" is entered in the input waiting state after execution of line 100, the data is separated into four blocks of 3-digit data, which are then assigned to the variables. The values of the variables are:

A=123, B=456, C=789, D=12

[Ex. 2.4.2-2]
Part of the data is assigned to variables.

```
50 INPUT USING 60 A$,B$,C$
60 FORMAT ␣␣###␣␣␣␣␣#####␣␣␣␣#
```

When the 20-character data "ABCDEFGHIJKLMNOPQRST" is entered in the input waiting state after execution of line 50, the spaces in the FORMAT statement serve as data separators and data read skip marks. Character data are assigned to the three variables as follows:

A$="CDE", B$="KLMNO", C$="T"

Input from Disk Files

Data can be read from a disk file or an external input device and assigned to a variable by specifying a logical device number in the INPUT USING statement. Refer to the explanation of the INPUT USING statement in "2.12.6 Other Input/Output" for details.

2.5 Output Instructions

2.5.1 PRINT Statement (Print)

Function

The PRINT statement displays data on the screen. Data can also be output to the printer, a disk file, or any other external output device by specifying the logical device number.

Format 1

```
PRINT␣[#{<1~9>                  },][{<String Expression>    }][{,}]......
        {<Arithmetic Variable>  } 1)  {<Arithmetic Expression>}  {;}
                                      {<Sub-instruction>      }
```

Format 2

```
<Sub-instruction> = {FEED(<Arithmetic Expression>)                              }
                    {SPACE(<Arithmetic Expression>)                             }
                    {TAB(<Arithmetic Expression>)                               }
                    {% HOME                                                     }
                    {% CURSOR(<Arithmetic Expression>, <Arithmetic Expression>) }
```

Note 1: The comma (,) at the end of a statement can be omitted.

Explanation

The PRINT statement displays characters and numbers on a display unit or outputs them to the printer. The contents of the variable or character string specified in the operand are output. Output is displayed on the screen if a logical device number is not specified. The form of output can be specified using symbols and sub-instructions in the operand.

Unless otherwise specified, output to the screen or printer by the PRINT statement starts at the current cursor or print head position. (Immediately after program execution starts, the cursor is positioned at the beginning of the line following the program execution command line.)

The function of the PRINT statement is to output the contents specified in the operand to an output device (like the display unit) in ASCII code. For example, a display unit and printer have a different code table to control them, so that a different action is performed even if the PRINT statement with the same output code is executed.

The form of data output varies depending on whether the data is numeric or character as shown below.

Numeric data..... Output is performed in the floating form when the value is within the range: $1 \times 10^{-4} \leqq |x| < 1 \times 10^{12}$ (e.g. 103.45).
Otherwise output is performed in the E form (e.g. 1.52E13). In both cases, one space is assigned at the beginning of output and data is left justified.

Character data.... Characters and symbols are output with left justification.

Commas (,) and semicolons (;) in the operand have the following meaning:

Semicolon (;) .... Used for continuous output.

Comma (,) ...... Used for output in the 20-column zone (described later).

The CR code ($0D_H$)[1] and the LF code ($0A_H$) are output automatically, and a new line is started if a comma or semi-colon is not specified at the end of a statement.

Note 1: ASCII code in 2-digit hexadecimal figures.

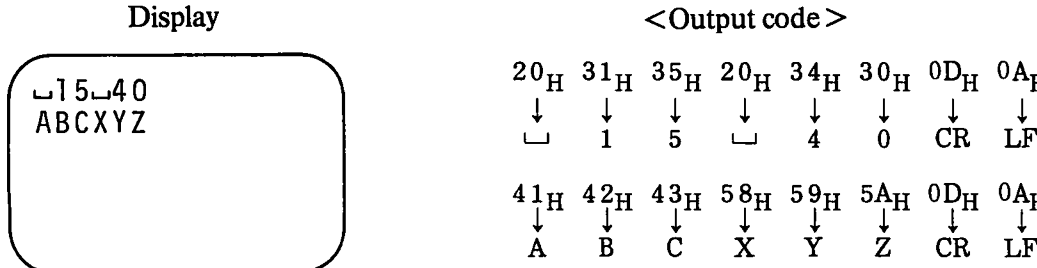
[Ex. 2.5.1-1]

```
40 A=15:B=40:X$="ABC":Y$="XYZ"
50 PRINT A;B
60 PRINT X$;Y$
```

In this example, after data are assigned to the variables on line 40, their contents are output by the PRINT statements on lines 50 and 60.

The display is shown below. (The cursor is directly below "A" on the display after execution of line 60.)

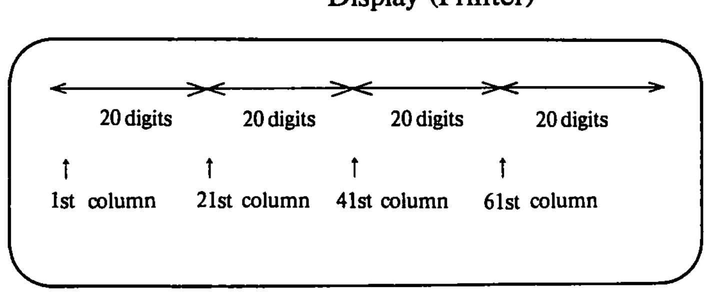
The ASCII code output is also shown for reference.



20-column Zone

The 20-column zone is an area consisting of 20 columns set by the PRINT statement in relation to the display and the printer. Characters in each 20-digit zone can be output by specifying a comma (,) in the operand of the PRINT statement.

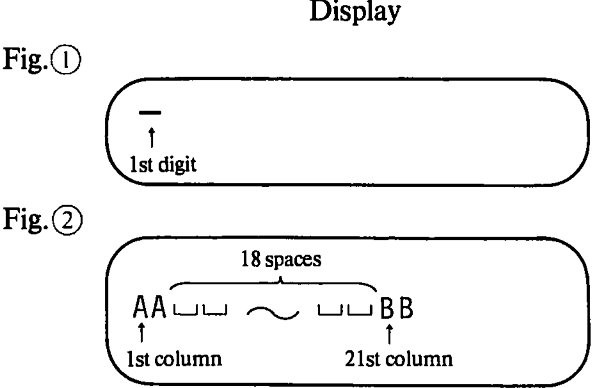
Display (Printer)



The comma in the operand specifies a shift of the cursor or print head to the start of the next 20-column zone. The PRINT statement itself counts the number of characters output and automatically outputs the number of spaces necessary to move the cursor or print head to the beginning of the next 20-column zone.

For example, when the cursor is at positon ① and `PRINT "AA", "BB"` is executed, the display is as shown in ②.

Display



The 20-column zone is valid for only one line on the display or the printer. When automatic line feed is performed (described later) and output is continued on the next line, the position of the 20-column zone may be shifted.

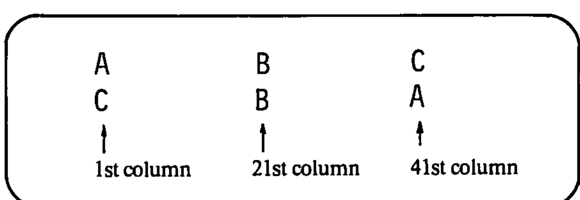
[Ex. 2.5.1-2]

Characters are displayed using a 20-column zone specification.

```
70  A$="A":B$="B":C$="C"
80  PRINT A$,B$,C$
90  PRINT C$,B$,A$
```

In this program example, output by the PRINT statements on lines 80 and 90 is as follows:

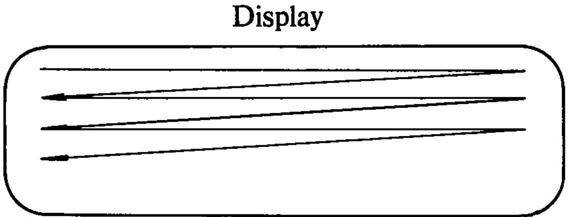
Display



Automatic Linefeed Function and Scrolling

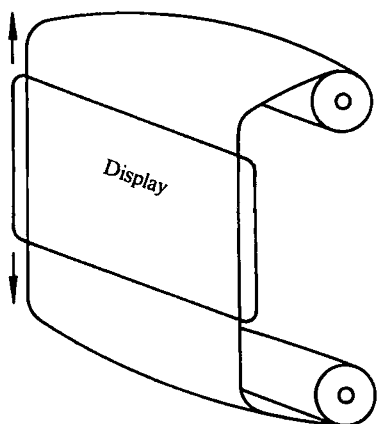
When output exceeds the line capacity of the display or printer, linefeed is performed automatically at the end of each line and output continues from the beginning of the next line.

Display

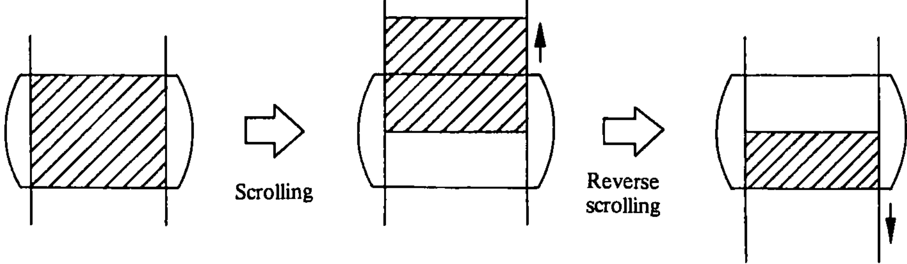


When linefeed is performed at the bottom line on the display screen, the screen contents move up to create new lines for output. When reverse linefeed is specified by the sub-instruction FEED (described later) the contents of the screen move down. These actions are called scrolling. The concept of scrolling is illustrated next.

Concept of Scrolling



Note Display contents that are scrolled off the screen are not reproduced even if the line is redisplayed on the screen by reverse scrolling.



Specification of Output Characters

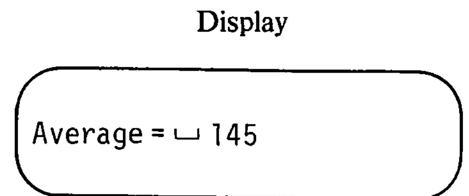
A string variable is usually specified in the operand for character output. Character output can be specified directly by specifying the character string between quotation marks like in the LET statement.

[Ex. 2.5.1-3]

```
70  A=145
80  PRINT "Average=";A
```

This program example produces the following display.

Display



Note Specify two quotation marks (" ") when a quotation mark will be output as a character.

PRINT "A""B" → Display: A"B

[Ex. 2.5.1-4]
Data is displayed on the screen using the function of the PRINT statement just described.

```
140 NAME1$="HARDIN":NAME2E$="AMES"
150 A=75:B=82
160 PRINT "PUPIL ";NAME1$;" 'S SCORE",A;"MARKS"
170 PRINT "PUPIL ";NAME2$;" 'S SCORE",B;"MARKS"
```

Executing this program example results in the following display.

Display

```
PUPIL␣HARDIN␣'S␣SCORE          ␣75MARKS
PUPIL␣AMES␣'S␣SCORE            ␣82MARKS
↑                              ↑
1st column                     21st column
```

Sub-instructions

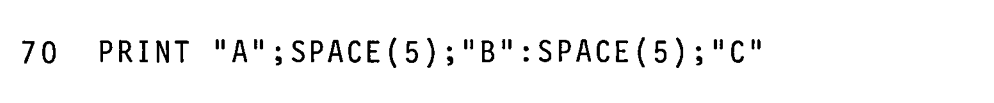
The following sub-instructions can be specified in the operand of the PRINT statement.

a) SPACE (<Arithmetic Expression>)

A specified number of spaces are output. The number of spaces that will be output is specified by an arithmetic expression or numeric value in parentheses. The value that will be specified must be a integer within the range:
$-255 \leqq x \leqq 255$.

When a negative value is specified, a backspace code (08_H) is output and the opposite action is performed (the output position moves one column to the left). When the cursor is located at the beginning of the line and the backspace code is output, the cursor moves to the end of the preceding line, but scrolling is not performed. When the destination of movement is outside of the current screen, the output position is the far-left column on the top line of the screen.

[Ex. 2.5.1-5]



```
70  PRINT "A";SPACE(5);"B":SPACE(5);"C"
```

Display

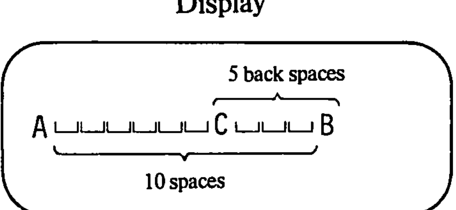
A␣␣␣␣␣B␣␣␣␣␣C

5 spaces   5 spaces

[Ex. 2.5.1-6]

```
80  PRINT"A";SPACE(10);"B";SPACE(-5);"C"
```

Display



b) FEED (<Arithmetic Expression>)

The number of lines specified are fed. The horizontal output position does not change.

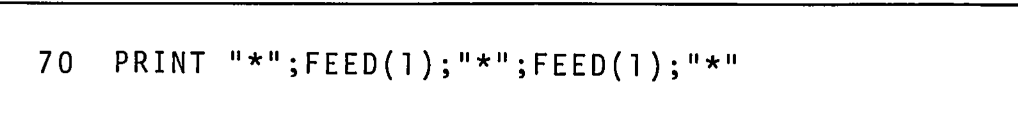
The number of lines that will be fed is specified by an arithmetic expression or numeric value in parentheses. The numeric value that will be specified must be an integer within the range: $-255 \leqq x \leqq 255$. When a negative value is specified, reverse linefeed is performed.

When the output position specified by FEED is outside the current screen, scrolling or reverse scrolling is performed.

When a positive value is specified, linefeed code ($0A_H$) is output. When a negative value is specified, a control code ($1B_H$, $4D_H$)[1] is output.

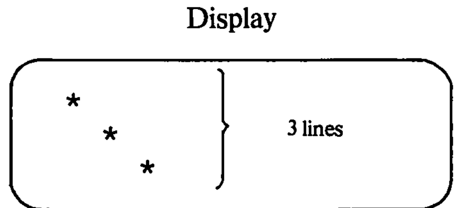
Note 1: the control code is only valid for display.

[Ex. 2.5.1-7]

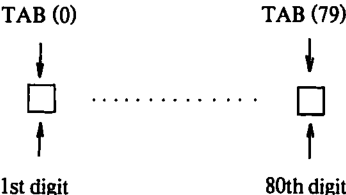


```
70  PRINT "*";FEED(1);"*";FEED(1);"*"
```

Display



c) TAB (<Arithmetic Expression>)



The output position moves to the specified column on the line. The column is specified by an arithmetic expression or a numeric value in parentheses. In column specification, the head of the line is 0. Integers within the range:
$0 \leqq x \leqq 255$ can be specified. When the value specified exceeds the length of a line (79 for the display), the movement continues on the next line.

The PRINT statement counts the number of the characters output. When the movement of the output position is specified by TAB, the PRINT statement calculates the number of spaces or backspaces required to produce output at the specified position and then outputs the necessary number of spaces (20_H) or backspaces (08_H).

[Ex. 2.5.1-8]

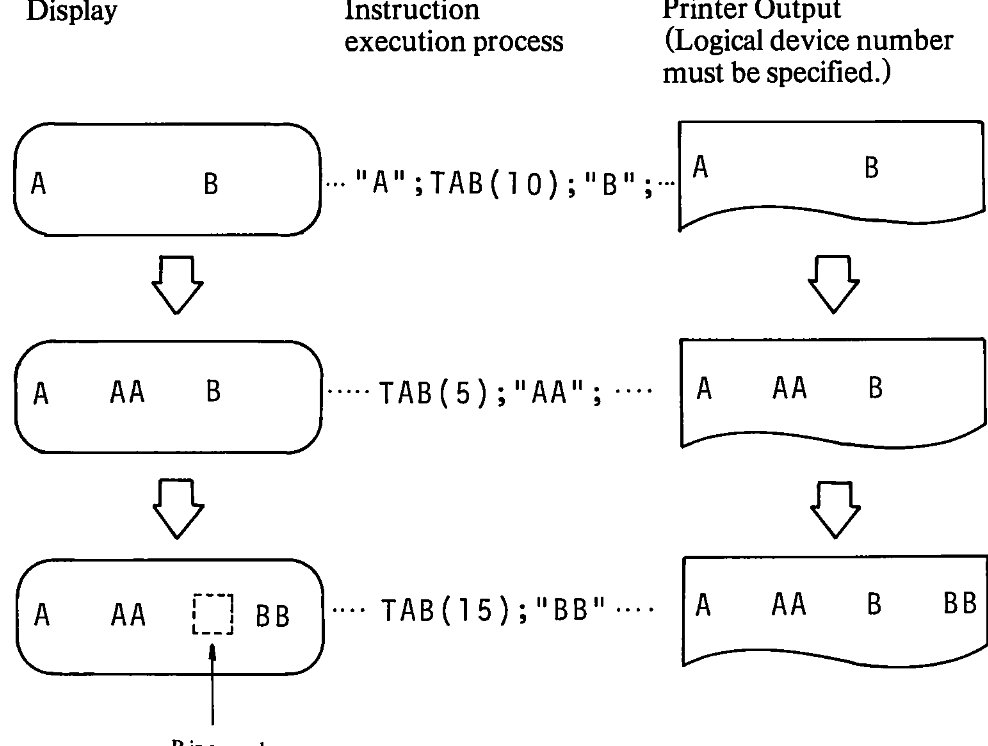
```
70  PRINT TAB(10);"ABC";TAB(20);"XYZ"
```

Display

```
ABC                XYZ
 ↑                  ↑
11th column      21st column
```

Note The movement of the output position using TAB is actually performed by outputting spaces or backspaces. Consequently the result of the PRINT statement execution is:

```
PRINT "A";TAB(10);"B";TAB(5);"AA";TAB(15);"BB"
```



d) %HOME

The entire screen contents are erased and the starting output position is set at the home position (first column of first line on the display).

Advice

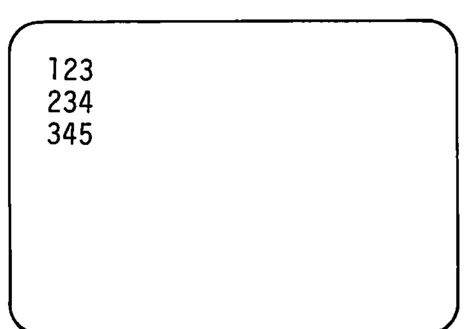
When program execution starts, the starting output positon on the display is the first column of the line immediately following the program execution command line. So, when output is performed to the display, the output resulting from program execution is performed after the display of command operations, etc. Execute "PRINT %HOME" to clear the display contents before executing any output instructions to the display.

PRINT %HOME not specified

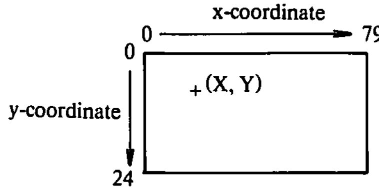
```
100END
%E
$SAVE ABC
SAVE TO A:ABC.BAS(Y/N)?Y
$RUN
123
234
345
```

Display by program execution (123, 234, 345)

PRINT %HOME specified



e) %CURSOR (<Arithmetic Expression>, <Arithmetic Expression>)



Coordinates on the display are specified and the starting output position is moved to that position.

x-coordinates 0~79 (horizontal) and y-coordinates 0~24 (vertical) are defined for the display. The coordinates are specified as (x, y) to indicate the starting output position.

The x and y coordinates are specified by arithmetic expressions or numeric values. Integers within the range: $-32767 \leqq x$ and $y \leqq 32767$ can be specified. Negative values are regarded as 0. Values exceeding 79 in x-coordinate specification are regarded as 79. Values exceeding 24 in y-coordinate specification are regarded as 24.

Note Coordinates are defined on the screen. So even if output is performed to coordinates (10,10), the result of output is moved to coordinates $(10,10-n)$ after the display contents are scrolled up.

[Ex. 2.5.1-9]

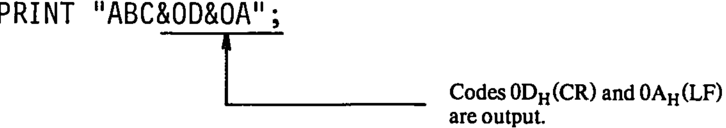
```
90  PRINT %CURSOR(5,10);"ABC";%CURSOR(10,10);"XYZ"
```

Display

```
ABC          XYZ
↑            ↑
(5,10)       (10,10)
```

Code Output

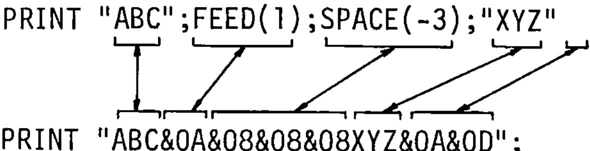
It is possible to specify the output of an ASCII code by specifying a hexadecimal figure in the PRINT statement. Output specification is performed by specifying a 2-digit hexadecimal code prefixed by the symbol "&" and enclosed with quotation marks (").



The codes used in the PRINT statement and their functions are described below.

- LF code ($0A_H$)
  Linefeed · · ·
  The cursor or paper is fed one line. The horizontal position of the cursor or print head does not change. The function is the same as that of FEED(1).
- CR code ($0D_H$)
  Carriage Return · · ·
  The cursor or print head returns to the first column of the current line. The vertical position of the cursor or print head does not change.
- BS code (08_H)
  Backspace · · ·
  The cursor is shifted one column to the left. The function is the same as that of SPACE (−1).
- BEL code (07_H)
  Bell · · ·
  The buzzer sounds for approximately 0.3 seconds.

Executing the following two PRINT statements produces the same result.



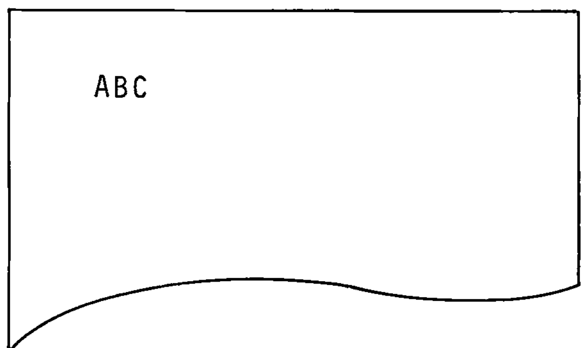
Output to Printer

Data can be output to a printer or other external output device by specifying a logical device number in the PRINT statement. The logical device number must be defined to the connecter where the output device is connected before executing the PRINT statement.

[Ex. 2.5.1-10]
Output is performed to the printer connected to connector no. 1.

```
40 OPEN #1,"LPT:"
50 PRINT #1,"ABC"
60 CLOSE #1
```

The following is printed by executing this program example.



Drawing Lines

The display characters for the AS-100 include special characters to draw lines. Lines can be drawn on the display by specifying the output of these special characters in the PRINT statement.

See "Appendix1 Character Codes" for the shapes of the special characters.

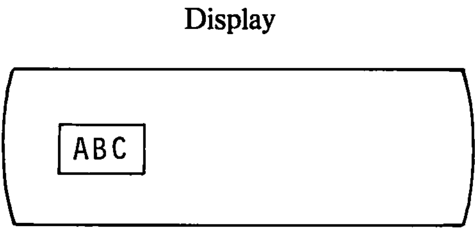
[Ex. 2.5.1-11]

Lines are drawn using the special characters "┌"($F0_H$), "—"($F5_H$), "┐"($F3_H$), "|"($F4_H$), "└"($F1_H$), and "┘"($F2_H$).

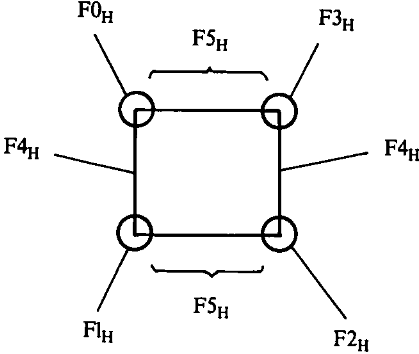
```
200 PRINT "&F0&F5&F5&F5&F3"
210 PRINT "&F4ABC&F4"
220 PRINT "&F1&F5&F5&F5&F2"
```

The following lines and characters are displayed on the screen when this program example is executed.

Display



The relationship between lines and special characters is shown below.



Output of Calculation Results

The result of a calculation can be output by specifying an arithmetic expression in the operand of the PRINT statement.

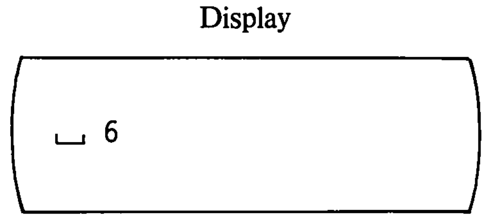
[Ex. 2.5.1-12]

The calculation result of an expression consisting of the three variables A, B, and C is output.

```
140 PRINT (A+B)/C
```

Assuming that A = 10, B = 20, and C = 5, the result of the above calculation (6) is output as follows:

Display



Output to Disk Files

Data can be output to a disk file or an external output device by specifying the logical device number in the PRINT statement. Refer to the explanation of the PRINT statement in "2.12.6 Other Input/Output" for details.

PRINT USING

FORMAT

2.5.2 PRINT USING Statement (Print Using)
FORMAT Statement (Format)

Function

These statements are used to output tables, etc., when the output form must be specified in detail.

Format

PRINT␣[#{<1~9> | <Arithmetic Variable>},]USING␣{<Line No.> | [<Label>]}␣{<Arithmatic Expression> | <Character Expression>}[,]···

FORMAT␣<Format Specification>

Explanation

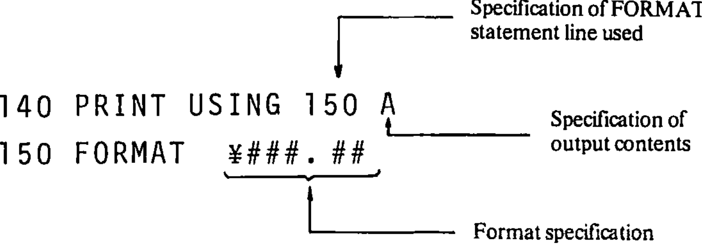
The contents of output are specified by the PRINT USING statement and the output form is specified by the corresponding FORMAT statement.

The PRINT USING statement and the FORMAT statement must be used together. Executing the FORMAT statement by itself has no effect.

Output Specification by the FORMAT statement

Several different format specification characters can be used to specify the output form in the FORMAT statement used with the PRINT USING statement. So do not use a common FORMAT statement for the INPUT USING statement and the PRINT USING statement.

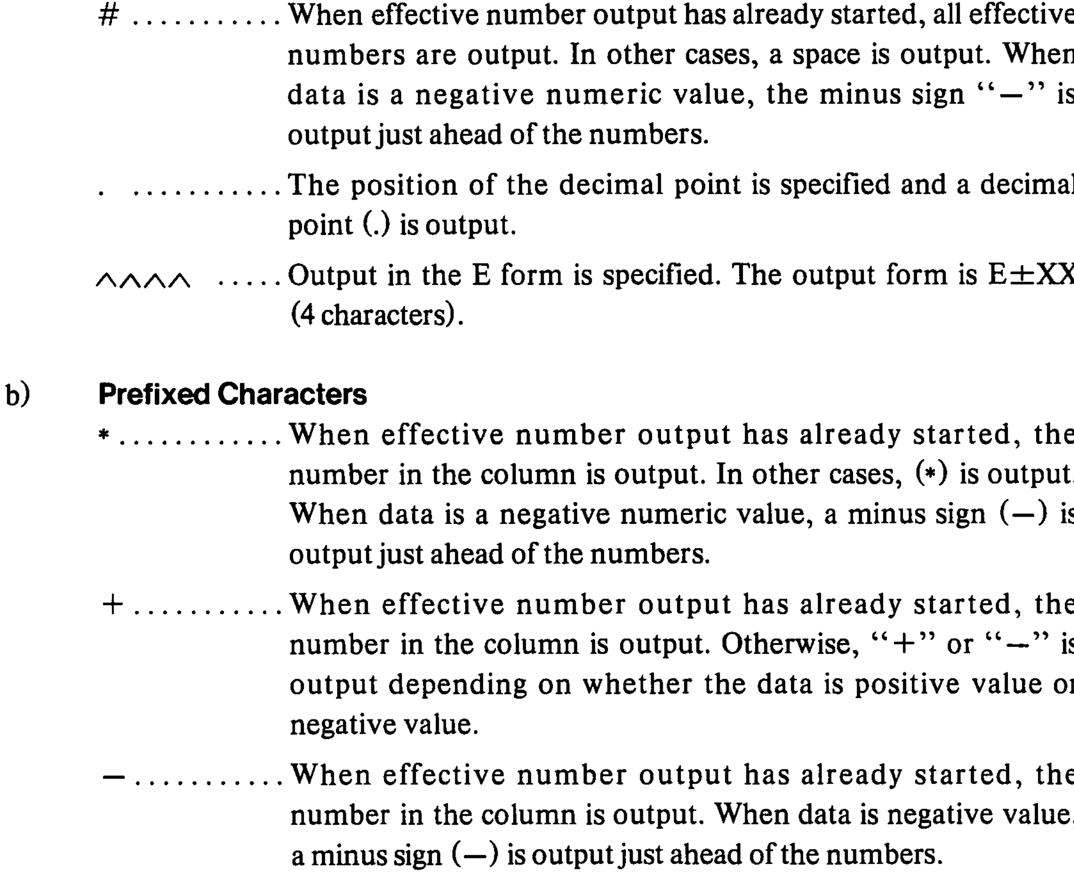
The relationship between the PRINT USING statement and the FORMAT statement is shown below.



In the FORMAT statement, the specification of the output format starts after a space is entered following the keyword. The specification of the output format by the PRINT USING statement is made only in the format. Except for the specification of the FORMAT statement line, etc., only the variables specifying the output contents and commas (separators) can be specified in the operand of the PRINT USING statement. (Commas used in this case do not specify printing in a 20-column zone.)

The following format specification characters can be specified in the operand of the FORMAT statement.

a) **Basic Characters**



.... When effective number output has already started, all effective numbers are output. In other cases, a space is output. When data is a negative numeric value, the minus sign "−" is output just ahead of the numbers.

. ... The position of the decimal point is specified and a decimal point (.) is output.

^^^^ ..... Output in the E form is specified. The output form is E±XX (4 characters).

b) **Prefixed Characters**

* .... When effective number output has already started, the number in the column is output. In other cases, (*) is output. When data is a negative numeric value, a minus sign (−) is output just ahead of the numbers.

+ .... When effective number output has already started, the number in the column is output. Otherwise, "+" or "−" is output depending on whether the data is positive value or negative value.

− .... When effective number output has already started, the number in the column is output. When data is negative value, a minus sign (−) is output just ahead of the numbers.

$ .... When effective number output has already started, the number in the column is output. The symbol "$" is output just ahead of the numbers.

0 .... When effective number output has already started, the number in the column is output. Otherwise, "0" is output. When data is a negative value, "−" is output at just ahead of the numbers.

c) **Suffixed Characters**

\+ .... "+" or "−" is suffixed to the data depending on whether the data is positive or negative.

− .... When data is a negative value, "−" is suffixed. If data is positive, a space is output.

d) **Inserted Characters**

, or ' .... When effective number output has already started, a comma (,) or an apostrophe (') is inserted at the position specified.

e) **Comment Characters**

Characters other than those just listed can be specified directly in the format as a comment.

Like in the INPUT USING statement, if there are fewer formats than variables that will be output, the formats are used repeatedly.

The relationship between format specification, output data, and output results are shown below.

Integer-type output

<Format>	<Data>	<Output>
#####	25	␣␣␣25
#####	−30	␣␣−30
#####	1.95	␣␣␣␣1 *The fractional part is truncated.
#####	1234567	##### *If output data exceeds the number of digits in the format, format specification is output as it is. This applies to all formats.

Decimal number-type output

<Format>	<Data>	<Output>
####.##	20	␣␣20.00
####.##	−0.1385	␣␣−0.13
####.##	12345.67	####.##

· E-type output

<Format>	<Data>	<Output>
##.###	1000	␣1.000E+03
##.###	−0.001234	−1.234E−03

Prefixed character output

<Format>	<Data>	<Output>
******	234	***234
******	−256	**−256
++++++	345	␣␣+345
++++++	−789	␣␣−789
++++++	23.45	␣␣␣+23
−−−−−−	789	␣␣␣789
−−−−−−	−795	␣␣−795
+#####	567	+␣␣567
+#####	−239	−␣␣239
$$$$$$	639	␣␣$639
$#####	329	$␣␣329
$*****	3276	¥*3276
000000	320	000320
000000	−603	−00603

Suffixed character output

<Format>	<Data>	<Output>
#####+	780	␣␣780+
#####+	−968	␣␣968−
#####−	824	␣␣824␣
#####−	−987	␣␣987−

PRINT USING
FORMAT

Inserted character output

<Format>	<Data>	<Output>
##,###	8726	␣8,726
##,###	23	␣␣␣␣23

Character data output

<Format>	<Data>	<Output>
#####	ABCDE	ABCDE
#####	ABC	ABC␣␣
#####	ABCDEFGH	ABCDE *Digits in excess of format specification are ignored for character data.

Output with comment

<Format>	<Data>	<Output>
###EXP ##	136,21	123EXP21
###EXP ##	ABCDEFGH	ABCEXPDE *A comment is regarded as a separator for numeric data. A comment is regarded simply as an inserted character for character data.

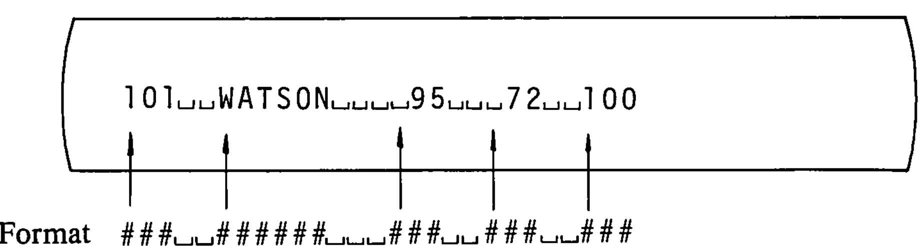
[Ex. 2.5.2-1]

Mixed output of numeric and character data.

```
100 PRINT USING 110 NO,NAME$,P1,P2,P3
110 FORMAT ###␣␣######␣␣␣###␣␣###␣␣###
```

Assuming that NO=101, NAME$="WATSON", P1=95, P2=72, and P3=100, the following is output.

Display

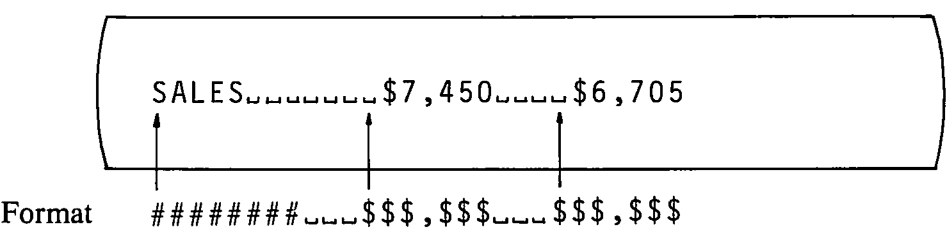


[Ex. 2.5.2-2]
A table of amounts is created using "$" and ",".

```
 90  PRINT USING 100 ITEM$,DNEW,DOLD
100  FORMAT ########␣␣␣$$$,$$$␣␣␣$$$,$$$
```

Assuming that ITEM$="SALES", DNEW=7450, and DOLD=6705, the following is output.

Display



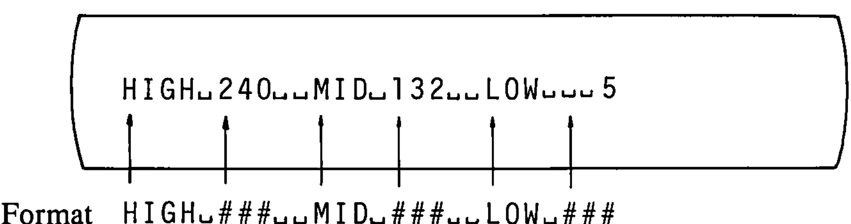
Format ########␣␣␣$$$,$$$␣␣␣$$$,$$$

[Ex. 2.5.2-3]
Output with direct specification of a comment in the FORMAT statement.

```
70  PRINT USING 80 D1,D2,D3
80  FORMAT HIGH␣###␣␣MID␣###␣␣LOW␣###
```

Assuming that D1 = 240, D2 = 132, and D3 = 5, the following is output.

Display



Format HIGH␣###␣␣MID␣###␣␣LOW␣###

PRINT USING

FORMAT

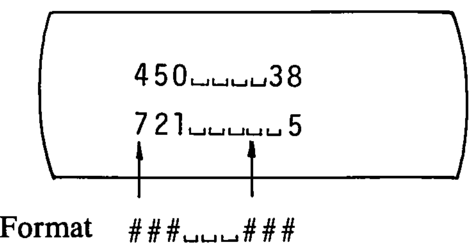
[Ex. 2.5.2-4]

A format is used repeatedly for a PRINT USING statement.

```
 90 PRINT USING 100 A,B,C,D
100 FORMAT ###␣␣␣###
```

Assuming that A=450, B=38, C=721, and D=5, the following is output.

Display



Format ###␣␣␣###

Output to disk files

Like with PRINT statement, data can be output to a disk file or an external output device by specifying the logical device number in the PRINT USING statement. Refer to the explanation of the PRINT statement in "2.12.6 Other Input/Output" for details.

2.6 Branch Instructions

2.6.1 GOTO Statement (Go To)

Function

The GOTO statement changes the program execution flow.

Format

```
GOTO ␣ { <Line No.> }
       { [<Label>]  }
```

Explanation

The GOTO statement is an instruction that changes the program execution flow. The line number or label must be specified in the operand. Specify number of the line to which the branch will be made. When the label is specified in the operand, the same label must be placed at the head of the line to which the branch will be made (immediately following the line number).

A branch destination is always the first statement in the line. A branch cannot be made to the second or following statement in a multi-statement line.

Note Alphabet letters and numbers can be used for labels. Enclose the label in square brackets ([ ]) and position it immediately following the line number. There is no limit to the number of characters in the label.

```
<Line No.> [<Label>]<Statement>
```

Only one branch destination label can be placed on a line.

An error occurs if the line contains only a label. Write a statement following the label.

[Ex. 2.6.1-1]

A branch to a specified line.

```
 :
70  PRINT X$
 :
300 GOTO 70
 :
```

Execution branches from line 300 to line 70.

[Ex. 2.6.1-2]
A branch to the line specified by the label.

```
 :
90  GOTO [CALC.ROUTINE]
 :
180 [CALC.ROUTINE]REM TOTALING
 :
```

Execution branches from line 90 to line 180.

[Ex. 2.6.1-3]
The GOTO statement is used as the second statement in a multi-statement line, and a branch is executed by the no-input operation.

```
 :
80  REM
 :
170 INPUT AGAIN:GOTO 80
180 REM
 :
```

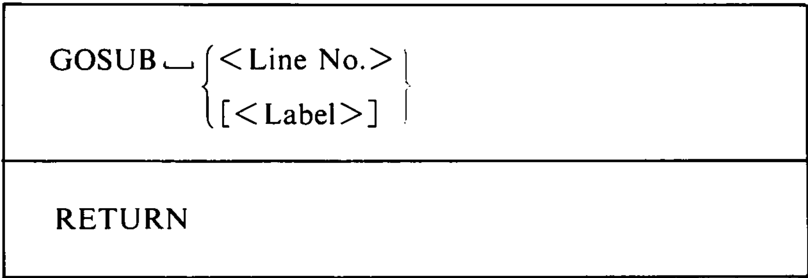
When data is input to the variable AGAIN on line 170, program execution branches to line 80 by the next statement, GOTO 80, and line 180 is executed when the no-input operation is performed.

2.6.2 GOSUB Statement (Go to Subroutine) RETURN Statement (Return)

Function

The GOSUB statement branches a program to a subroutine and executes the subroutine. The RETURN statement returns program execution to the main routine and executes the statement next to the GOSUB statement where the branch occurred.

Format

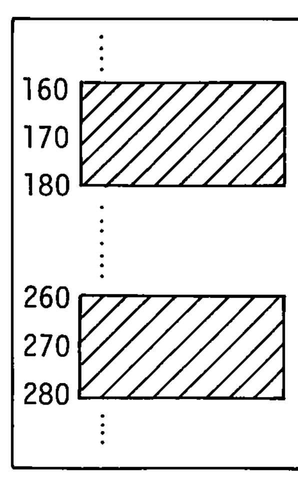


Explanation

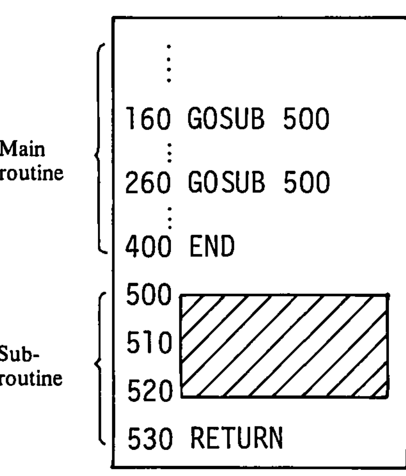
When there are several identical parts in a program, they are extracted from the program and made into a common routine [1]. This is called a subroutine. Using subroutines makes a program easy to understand and decreases the program volume.

The GOSUB statement branches program execution to a subroutine and executes it. The RETURN statement returns execution from the subroutine to the original or main routine.

[Program without subroutine]



[Program with subroutine]



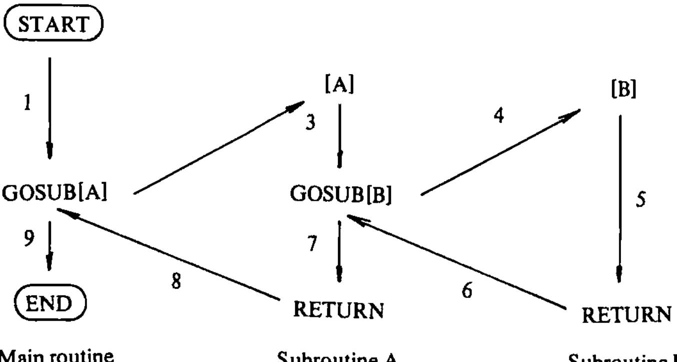
Note 1: A routine is a part of a program that performs a particular job.

GOSUB
RETURN

A line number or a label can be specified in the operand of the GOSUB statement. Specify the line number on the first line of a subroutine. When a label is specified in the operand, the same label must be placed at the beginning of the line where the subroutine starts (immediately following the line number). Program execution proceeds to the statement next to the GOSUB statement to which the branch was made when the RETURN statement is executed.

One subroutine can be branched to another. This is called the nesting of a subroutine. Any number of subroutines can be nested as long as the capacity of the stack area in memory is sufficient. But because the stack area is also shared by the FOR and the NEXT statements (described later), there is a correlation to nesting. Fig. 2 shows the nesting order in two levels.

Fig. 2



[Ex. 2.2.6-1]

An input error warning is made into a subroutine.

When a value greater than 100 is input to variable X, the alarm sounds and "ERROR" is displayed.

```
140 INPUT X
150 IF X<=100 GOTO [Calculation A]
160 GOSUB [Error]
170 GOTO 140
 :
240 INPUT Y
250 IF Y<=100 GOTO [Calculation B]
260 GOSUB [Error]
270 GOTO 240
 :
500 [Error] REM Buzzer & Print
510 PRINT "&07";
520 PRINT "ERROR"
530 RETURN
```

Line 510 is the PRINT statement to activate the alarm.

2.6.3 IF Statement (If)

Function

A branch or statement is executed according to specified conditions.

Format

```
IF␣{<Conditional Expression>}␣{GOTO␣ {<Line No.> }}
   {<Arithmetic Expression> }  {      {[<Label>] }}
                               {THEN ␣ <Statement>  }
```

Explanation

This instruction changes the order of program execution depending on whether a specified condition is satisfied or whether the value of an expression is 0.

When a conditional expression is satisfied or when the value of an arithmetic expression is not 0, the GOTO statement or the statement following THEN is executed.

When a conditional expression is not satisfied or when the value of an arithmetic expression is 0, the statements on the next line are executed.

Six relational operators (<, >, =, ≠, ≧, and ≦) can be used in conditional expressions. In programs, they are indicated as follows:

< → <
> → >
= → =
≠ → <>
≦ → <=
≧ → >=

In judging the condition following IF, −1 means that the condition is satisfied and 0 means that it isn't satisfied. This processing method allows logical operations (logical product, logical sum, exclusive OR, and negation) to judge two or more conditions.

Logical operations are performed as follows:

Logical product (AND)

X	Y	Logical product of X and Y
0	0	0
0	−1	0
−1	0	0
−1	−1	−1

Logical sum (OR)

X	Y	Logical sum of X and Y
0	0	0
0	−1	−1
−1	0	−1
−1	−1	−1

Exclusive OR (XOR)

X	Y	Exclusive OR of X and Y
0	0	0
0	−1	−1
−1	0	−1
−1	−1	0

Negation (NOT)

X	Negation of X
0	−1
−1	0

The logical operators shown in parentheses are used in programs.

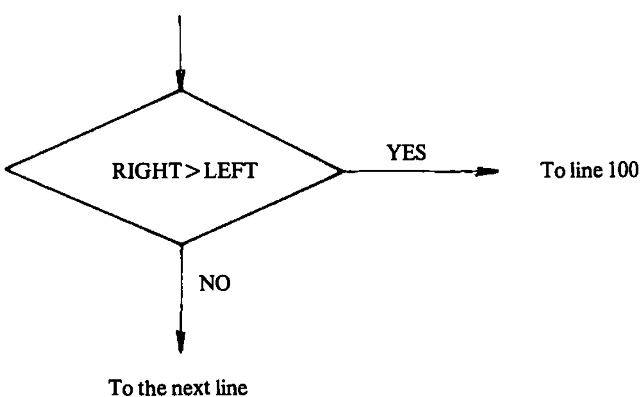
Note A GOTO or THEN statement following the IF statement must be written on the same line with the IF statement. An error occurs if the THEN statement which follows IF and the condition specified are written on the next line.

[Ex. 2.6.3-1]

The program branches according to a conditional expression comparing one variable with another. When the value of the variable RIGHT is greater than that of the variable LEFT, program execution branches to line 100.

```
 :
50  IF RIGHT>LEFT GOTO 100
 :
                    (Same with  THEN GOTO 100)
```

This is shown in the flowchart below.

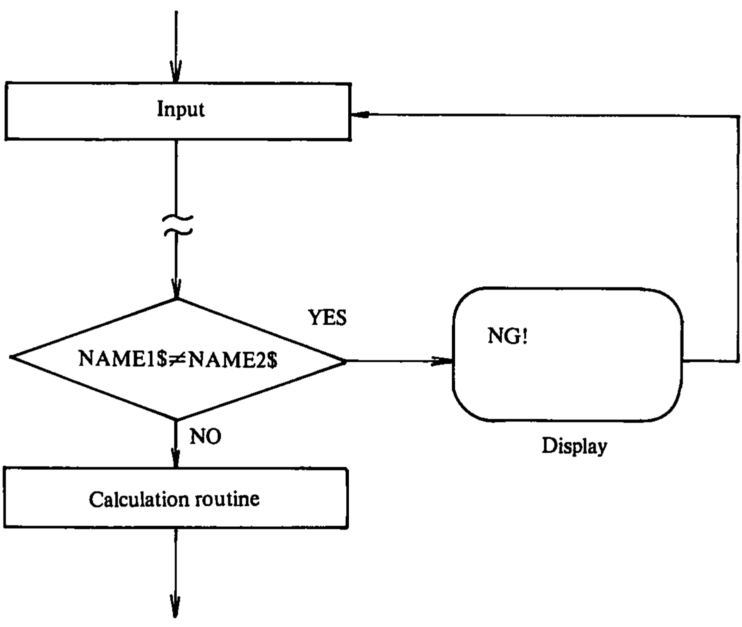


[Ex. 2.6.3-2]

When the comparison condition of string variables is satisfied, the statement following THEN is executed.

```
 :
80  [Input] REM Input Routine
 :
150 IF NAME1$<>NAME2$ THEN PRINT "NG!":GOTO [Input]
160 REM Calculation Routine
 :
```

If the values of string variables NAME1$ and NAME2$ are the same, execution proceeds to the next line. When they are not, "NG!" is displayed and program execution branches to the label [Input].



This example can be written as follows:

```
 :
80  [Input] REM  Input Routine
 :
150 IF NAME1$=NAME2$ GOTO 170
160 PRINT "NG!":GOTO [Input]
170 REM Calculation Routine
 :
```

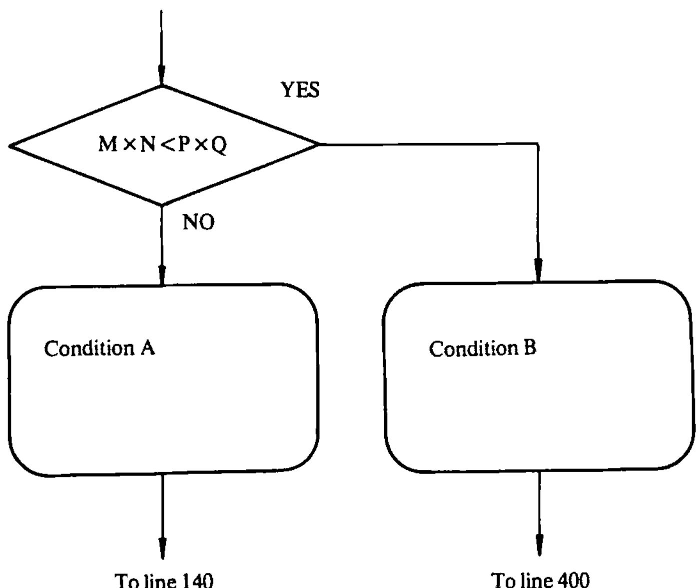
In this case, when the contents of NAME1$ and NAME2$ on line 150 are the same, execution branches to the calculation routine on line 170. When they are not, line 160 is executed and execution branches to line 80.

[Ex. 2.6.3-3]

The program branches according to the comparison of arithmetic expressions.

```
 :
120 IF M*N<P*Q THEN PRINT "Condition B" :GOTO 400
130 PRINT "Condition A"
140 REM Condition A
 :
400 REM Condition B
 :
```

When the values of variables M, N, P, and Q satisfy the condition $M \times N < P \times Q$, "Condition B" is displayed and program execution branches to line 400. Otherwise, "Condition A" is displayed and execution proceeds to the next line.

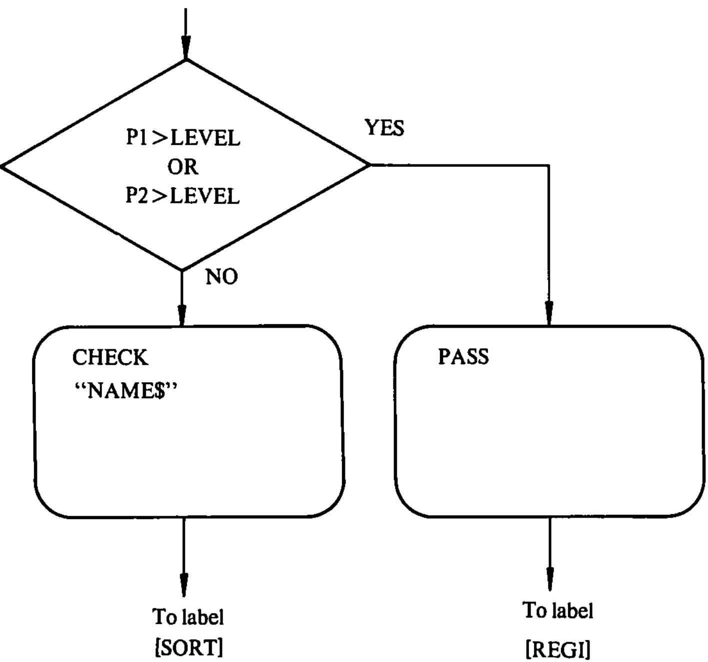


[Ex. 2.6.3-4]

Program execution branches according to the logical sum (OR) of the two conditional expressions.

```
 :
80  [SORT] REM  High-Low Comparison
 :
170 IF (P1>LEVEL)OR(P2>LEVEL) THEN PRINT "PASS": GOTO [REGI]
180 PRINT "CHECK&07";NAME$;GOTO [SORT]
 :
400 [REGI] REM NUMERICAL REGISTRATION
```

When the value of either variable P1 or P2 is greater than the value of variable LEVEL, "PASS" is displayed and program execution branches to the label [REGI]. Otherwise "CHECK" is displayed, the alarm sounds, the contents of string variable NAME$ are displayed, and program execution branches to the label [SORT].



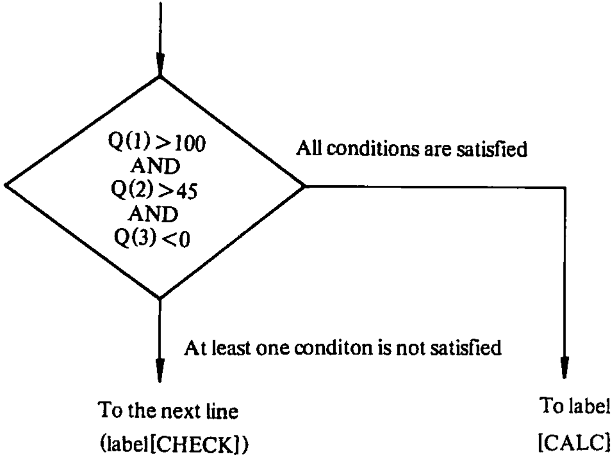
[Ex. 2.6.3-5]

Program execution branches according to the logical product (AND) of three conditional expressions.

When variables Q(1), Q(2), and Q(3) satisfy all of the following conditions, program execution branches to label [CALC]:

Q(1) >100, Q(2) >45, Q(3) <0

```
 :
150 IF (Q(1)>100)AND(Q(2)>45)AND(Q(3)<0) GOTO [CALC]
160 [CHECK] REM VALUE CHECK
 :
270 [CALC] REM CALCULATION OF VALUE
 :
```



Each conditional expression whose specified condition is satisfied has a value of −1. When the condition is not satisfied, each has a value of 0. If at least one of the conditional expressions has a value of 0, the total logical product is 0. Program execution branches to the label [CALC] only when all of the conditions are satisfied. When at least one of the conditions is not satisfied, execution proceeds to the next line.

2.6.4 ON Statement (On)

Function

Program execution branches according to the value of a variable or expression.

Format

```
ON␣<Arithmetic Expression>␣{GOTO  }␣{<Line No.>  } [ ,] ···
                           {GOSUB }  {[<Label>]  }
```

Explanation

The value of an arithmetic expression following ON is converted to an integer and program execution branches to the line indicated by the integer value. The relationship between the value of an arithmetic expression and the branch destination line is shown below.

ON <Arithmetic Expression> GOTO <Line No.1>, <Line No.2>, ..., <Line No.n>

When the value of the arithmetic expression is 1, program execution branches to the line specified in <Line No. 1>.

When the value of the arithmetic expression is 2, program execution branches to the line specified in <Line No. 2>.

⋮

When the value of the arithmetic expression is n, program execution branches to the line specified in <Line No. n>.

When the value of the arithmetic expression is 0 or negative, execution proceeds to the statement next to the ON statement.

When the value of the arithmetic expression is greater than the number of destinations, execution also proceeds to the next statement.

As described above, a line number or a branch destination label must be specified following the GOTO or GOSUB statement.

[Ex. 2.6.4-1]

When the value of variable CODE is 1, program execution branches to line 200. When the value is 2, program execution branches to line 350. When the value is 3, program execution branches to line 540.

```
 :
70  ON CODE GOTO 200,350,540
 :
200 REM CODE=1
 :
350 REM CODE=2
 :
540 REM CODE=3
 :
```

[Ex. 2.6.4-2]

Under the same conditions as in Ex. 2.6.4-1, program branches according to the label.

```
 :
70  ON CODE GOTO [B1],[B2],[B3]
 :
200 [B1] REM CODE=1
 :
350 [B2] REM CODE=2
 :
540 [B3] REM CODE=3
 :
```

[Ex. 2.6.4-3]

Program execution branches according to the conditional expression.

When one of the variables A, B, or C has positive value, program execution branches to line 250. When two of them have positive values, program execution branches to line 350. When all three variables have positive values, program execution branches to line 450.

```
 :
120 ON -((A>0)+(B>0)+(C>0)) GOTO 25,350,450
 :
250 REM 1 POSITIVE VALUE
 :
350 REM 2 POSITIVE VALUES
 :
450 REM 3 POSITIVE VALUES
 :
```

Each conditional expression has a value of −1 when its condition is satisfied and 0 if it is not. So the value of the expressions following ON is equal to the number of variables that have positive values.

2.7 Loop Instructions

2.7.1 FOR Statement (For) NEXT statement (Next)

Function

These instructions repeat the same processing.

Format

FOR␣<Arithmetic Variable>=<Arithmetic Expression 1>␣TO␣<Arithmetic Expression 2>[␣STEP<Arithmetic Expression 3>]
NEXT␣<Arithmetic Variable>

Explanation

A pair of FOR and NEXT statements are used to repeat all program lines from the FOR statement to the NEXT statement. This type of execution is called a loop. The arithmetic variable following FOR counts the number of times the loop is executed. The arithmetic variable in the operand of the NEXT statement must be the same as the arithmetic variable following FOR.

Arithmetic expressions 1~3 are specified to determine the number of loop repetitions. The value of <Arithmetic Expression 1> is called the starting value and specifies the arithmetic variable's initial value in the loop. The value of <Arithmetic Expression 2> is called the ending value and specifies the value of the arithmetic variable at which loop execution will end.

The value of <Arithmetic Expression 3> is called the increment value and specifies the amount added to the arithmetic variable each time the loop is executed.

Numeric values, simple variables, and array variables are used for arithmetic expressions 1~3. The increment value is added to the starting value with each loop execution. The loop ends when the value of the arithmetic variable is greater than the ending value. The statement following NEXT is executed when the loop ends.

<Arithmetic Expression 3> can be omitted. When it is omitted, 1 is automatically specified. Negative values can be used for arithmetic expressions 1~3.

The value of the arithmetic variable and the ending value are compared and the increment value is added by the NEXT statement. So, the loop is executed at least once no matter what the starting, ending, and increment values are.

The values of arithmetic variable and arithmetic expressions 1～3 can be assigned to other variables or referenced in a loop, but their values cannot be changed.

A branch can be performed during loop execution. A branch to a line in the middle of the loop cannot be performed and causes an error. If program execution branches during loop execution, the value of the arithmetic variable does not change.

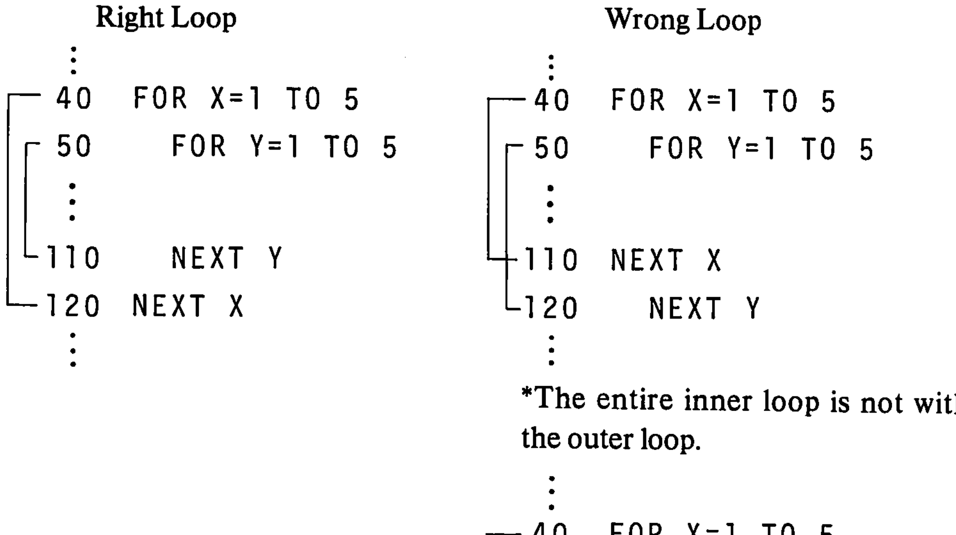
Loops between FOR and NEXT can be included in another loop between FOR and NEXT. This is called nesting.

```
┌── 60   FOR X=1 TO 5
│┌─ 70     FOR Y=1 TO 3
││   :
│└─110     NEXT Y
└──120   NEXT X
```

```
┌─── 60   FOR I=1 TO 5
│┌── 70     FOR J=1 TO 3
││┌─ 80       FOR K=1 TO 10
│││   :
││└─110       NEXT K
││┌─120       FOR L=1 TO 4
│││   :
││└─150       NEXT L
│└── 160    NEXT J
└─── 170  NEXT I
```

Any number of nesting levels can be specified as long as the capacity of the stack area in memory is sufficient. Because the GOSUB statement and the RETURN statement also use the stack area, there is a correlation with their nesting.

In nesting, the entire inner loop must be within the outer loop. The arithmetic variable specified for the outer loop must not be specified for the inner loop.



*The entire inner loop is not within the outer loop.

```
  :
40  FOR X=1 TO 5
50    FOR Y=1 TO 5
  :
110 NEXT X
  :
```

*Line 50 is ignored so there is no error, but FOR and NEXT are used incorrectly.

[Ex. 2.7.1-1]

The loop is executed five times with the condition that variable I = 1, 2, 3, 4, and 5. The value of the array variable WORK (I) is displayed. The increment value is omitted (1).

```
:
50  FOR I=1 TO 5
60    PRINT WORK(I)
70  NEXT I
:
```

When an loop execution ends, variable I has a value of 6.

[Ex. 2.7.1-2]

The loop is executed five times with the condition that variable COUNT = 1, 3, 5, 7, and 9. The value of COUNT is displayed.

```
  :
110  FOR COUNT=1 TO 10 STEP 2
120    PRINT COUNT
130  NEXT COUNT
  :
```

When loop execution ends, variable COUNT has a value of 11.

[Ex. 2.7.1-3]

The loop is executed five times with the condition that variable I = 0.2, −0.2, −0.6, −1.0, and −1.4. The value of 2 × I is displayed during loop execution.

```
  :
210  FOR I=0.2 TO -1.4 STEP -0.4
220    PRINT 2*I
230  NEXT I
  :
```

When loop execution ends, variable I has a value of −1.8

[Ex. 2.7.1-4]

This is an example of 2-level loop nesting. Data are input in sequence to array variables CONST (1, 1) ~ CONST (4, 4).

```
10  DIM CONST(4,4)
 :
310 FOR J=1 TO 4
320   FOR I=1 TO 4
330     INPUT CONST(I,J)
340   NEXT I
350 NEXT J
 :
```

When loop execution ends, variables I and J have values of 5.

[Ex. 2.7.1-5]

This is an example of an error.

It is impossible to branch program execution from line 100 to line 410.

```
 :
70  FOR J=0 TO 20
 :
100   IF A=0 GOTO 410 ---+  Wrong branch
 :                       |
200 NEXT J               |
 :                       |
400 FOR I=0 TO 5         |
410   FOR J=0 TO 3 <-----+
 :
500   NEXT J
510 NEXT I
 :
```

2.8 Constant Definition Instructions

2.8.1 READ Statement (Read)
DATA Statement (Data)

Function

Numeric values and characters are assigned to variables.

Format

READ ␣ <Variable> [,] ···
DATA ␣ { <Constant> / <Character> } [,] ···

Explanation

The READ statement specifies the variable to which data will be assigned and the DATA statement actually assigns the data.

The variable specified in the READ statement must correspond to the data specified in the DATA statement. So the number and the type of variables (arithmetic or string) must agree with corresponding data. But there is a special way to make the number of data smaller than that of variables to produce the same result as the no-input function of the INPUT statement.

Data assigned to a string variable does not need to be enclosed with quotation marks ("). However the data must be in quotes if quotation marks are used as data. Characters which cannot be entered through the keyboard can be specified by a hexadecimal code following "&".

READ
DATA

[Ex. 2.8.1-1]

Data are assigned to variables as follows:

10	20	30	WHITE	RED	40	50	60
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
X	Y	Z	A$	B$	C(1)	C(2)	C(3)

```
10  DIM C(3)
 :
150 READ X,Y,Z
 :
300 READ A$,B$
 :
350 FOR I=1 TO 3
360 READ C(I)
370 NEXT I
 :
400 DATA 10,20,30,WHITE
410 DATA RED,40,50,60
 :
```

Lines 400 and 410 can be combined into one line as DATA 10, 20, 30, WHITE, RED, 40, 50, 60.

[Ex. 2.8.1-2]

This is an example in which there are fewer variables than data (no input).

Only two data are specified in the DATA statement on line 100.

When the value of N entered on line 20 is less than or equal to 2, the data on line 100 is read and program execution proceeds to line 60. When the value of N is greater than 2, execution proceeds to the next line (line 50), "NO DATA" is displayed and program execution ends.

```
10  DIM A(10)
20  INPUT N
30  FOR I=1 TO N
40  READ A(I):GOTO 60
50  PRINT "NO DATA":END
60  NEXT I
 :
100 DATA 11,22
 :
```

2.8.2 RESTORE Statement (Restore)

Function

This instruction makes the first data in the DATA statement correspond to the variable specified in the READ statement that immediately follows.

Format

```
RESTORE
```

Explanation

There must be a one to one correspondence between the number of variables in the READ statement and the number of data in the DATA statement. But when the processing requires that the same data are read in the same order, the correspondence can be changed using the RESTORE statement. The first data in the DATA statement is made to correspond to the variable of the READ statement that immediately follows the RESTORE statement.

[Ex. 2.8.2-1]

Using the READ statement, data 10, 20, and 30 are assigned so that X=10, Y=20, Z=30, A=10, and B=20.

```
 :
30   READ X,Y,Z
40   RESTORE
50   READ A,B
 :
100  DATA 10,20,30
 :
```

2.9 Program Control Instructions

2.9.1 END Statement (End)

Function

This instruction ends program execution.

Format

Explanation

Several END statements can be placed on various program lines according to the program flow. The program ends when the END statement is executed.

The END statement in a subprogram (i.e. a program called by the CALL statement) has the same function as the RETURN statement in a subroutine. When the END statement is executed in a subprogram, program execution returns to the main program and the statement following the CALL statement is executed.

[Ex. 2.9.1-1]

Program execution ends after a branch by the IF statement.

```
 :
200 IF A>0 GOTO 300
210 REM CASE OF A=<0
 :
290 END
300 REM CASE OF A>0
 :
380 END
```

2.9.2 BYE Statement (Bye)

Function

This statement ends BASIC and returns the system to the OS mode.

Format

BYE

Explanation

The BYE statement ends the program and returns the system to the OS mode. The function is exactly the same as that of the BYE command. It is used like the END statement.

[Ex. 2.9.2-1]

```
 :
400    BYE
```

In the above example, BASIC program execution ends and the system returns to the OS mode.

Note

When BASIC program execution is started by a BASIC command in a SUBMIT file (see the "CP/M-86 User's Manual"), the next line in the SUBMIT file is executed after the BASIC program is ended using the BYE statement.

2.10 Function Definition Statement

2.10.1 DEF FN Statement (Define Function)

Function

This instruction defines an expression used repeatedly as a user defined function.

Format

```
DEF␣FN<Function Name>(<Variable>[,]···)=<Defined Expressions>
```

Explanation

When the same calculation is repeated many times in a program, the calculation expression can be defined as a function using this instruction.

The function name is specified according to the same rules as those used for variable name. The sub-keyword FN must be placed in front of the function name. "FN" must always be specified in capital letters.

Two or more variables can be specified in an expression. This statement must be executed prior to execution of the statement that uses the function.

[Ex. 2.10.1-1]
Coshx = $(e^x + e^{-x})/2$ is defined as FNCOSH(X).

```
 :
10   DEF FNCOSH(X)=(EXP(X)+EXP(-X))/2
 :
150  FOR I=1 TO 360
 :
250    LET P=FNCOSH(I)*10
260    PRINT P
270  NEXT I
 :
```

EXP(X) is one of the built-in functions explained later.

DEF FN

[Ex. 2.10.1-2]
$2X^2+4Y-Z$ is defined as FNA (X, Y, Z).

```
 :
10   DIM D(3)
20   DEF FNA(X, Y, Z)=2*X**2+4*Y-Z
 :
150  LET ANS=FNA(D(1),D(2),D(3))
 :
```

The following calculation is performed on line 150.
$ANS=2\times(D(1))^2+4\times D(2)-D(3)$
Assuming that D(1)=3, D(2)=2, and D(3)=1,
$ANS=2\times 3^2+4\times 2-1=25$.

2.11 Program Call Instructions

2.11.1 CALL Statement (Call)
PARAM Statement (Parameter)

Function

The CALL statement loads other BASIC programs stored on the disk (called subprograms) and executes them.

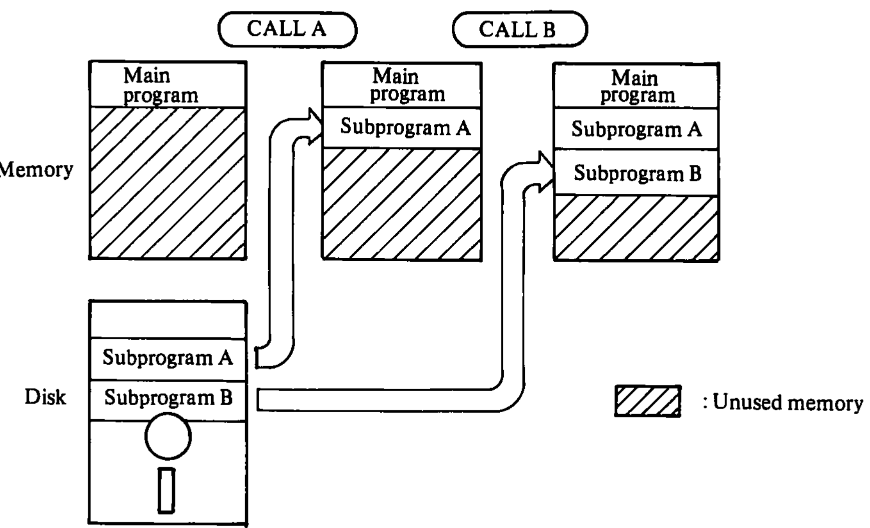
The PARAM statement shares data between the main program and subprograms.

Format

CALL␣<Program>[(<Variable>[,] ···)]
PARAM␣<Variable>[,] ···

Explanation

The CALL statement reads a subprogram from the disk, loads it to memory, and then executes it.



So when the CALL statement is executed, an error occurs unless there is sufficient memory to store the subprogram.

CALL
PARAM

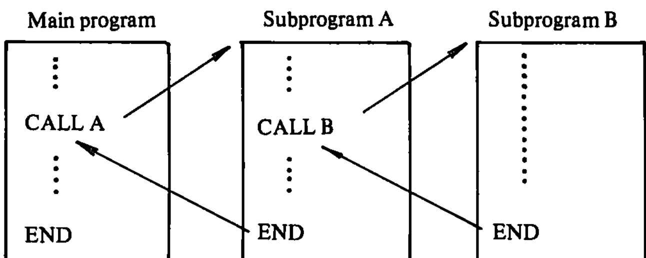
Immediately after the CALL statement is executed, the system confirms that the specified subprogram is in memory. If found, the subprogram is executed. If not, the subprogram is searched for on the disks, starting with the disk in drive A. If the subprogram is found, it is loaded to memory for execution. An error occurs if the subprogram is not found.

Data can be shared with the subprogram by specifying a variable after the subprogram name of the CALL statement, and then placing the PARAM statement in which the variables are specified at the beginning of the subprogram. Another statement cannot be written before the PARAM statement.

If the END statement is executed in the subprogram, the statement next to the CALL statement in the main program is executed. The subprogram remains in memory after execution. Even if the same subprogram is called, it is not read from the disk again. The FREE statement (described later) can be used to prevent a memory overflow when subprograms are called.

The subprogram starting with a PARAM statement cannot be executed independently. It must be called by the CALL statement.

The CALL statement can be executed to call a different subprogram in a subprogram. The nesting of CALL statements is the same as the nesting of GOSUB statements.



[Ex. 2.11.1-1]

Variable D and data NAME$(1) ~ NAME$(3) are shared between program A (main program) and program B (subprogram).

```
 5  REM PROGRAM A
10  DIM NAME$(3)
 :
110 PRINT D,NAME$(1)
120 CALL B(D,NAME$(*))
130 PRINT D,NAME$(1)
 :
```

```
10  PARAM D,NAME$(*)
15  REM PROGRAM B
 :
```

Asterisk (*) indicates all array variables.

The contents of variable D and data NAME$(1) ~ NAME$(3) are shared by the two programs using the CALL statement on line 120 in program A and the PARAM statement on line 10 in program B.

The contents displayed on line 110 are shared with program B which is called by the CALL statement on line 120. There is a statement in program B to update the contents of variable D and NAME$(1) ~ NAME$(3). After execution of program B, the updated data of each variable are shared with program A.

Note

As the example shows, the array variables shared by programs A and B using the PARAM statement do not need to be defined by the DIM statement in program B. But array variables not specified by the PARAM statement, even if the same name is defined in main program, must be defined in program B by the DIM statement. In this case, the contents of the variables in programs A and B are not related. They are treated as different variables.

CALL
PARAM

[Ex. 2.11.1-2]

Data are assigned directly to the variables in the subprogram called.

Instead of making a variable correspond to a variable using the CALL and PARAM statements, the data specified in the CALL statement are assigned directly to the subprogram called.

```
10  REM PROGRAM A
 :
100 CALL B(100,X)
 :
```

```
10  PARAM A,B
20  REM PROGRAM B
 :
```

When execution proceeds to program B, the contents of variables A and B are A = 100 and B = (Value of X in program A). When execution returns to program A, only the value of B is given to X.

Numeric values and characters can be specified directly in the CALL statement.

[Ex. 2.11.1-3]

An expression is specified in the CALL statement.

```
10  REM PROGRAM A
 :
100 CALL B(X,Y+Z)
 :
```

```
10  PARAM A,B
15  REM PROGRAM B
 :
```

When execution proceeds to program B, the contents of variables A and B are A = (Value of X) and B = (Value of Y + Z). When execution returns to program A, the contents of A are assigned to X in program A but the contents of Y and Z do not change.

[Ex. 2.11.1-4]

A nesting of program call is executed.

Program C is called in program B which is called in program A.

```
10  REM PROGRAM A
 :
100 CALL B(X,Y)
 :
```

Program B

```
10  PARAM X,Y
15  REM PROGRAM B
 :
200 CALL C(Z)
 :
500 END
```

Program C

```
10  PARAM Z
15  REM PROGRAM C
 :
300 END
```

2.11.2 FREE Statement (Free)

Function

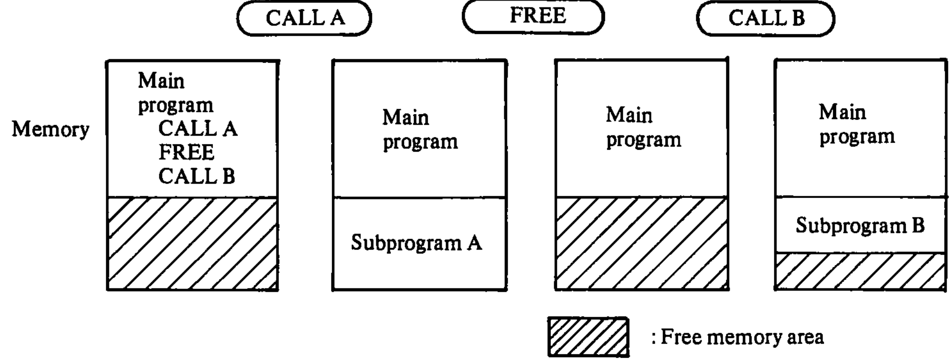
This instruction deletes a subprogram from memory.

Format

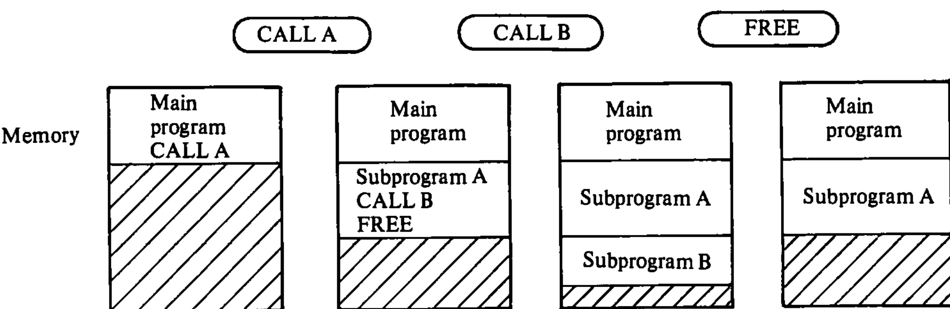
```
FREE
```

Explanation

The subprogram called remains in memory after execution, so the memory may overflow when several subprograms are called, even if some programs are not used. These unnecessary subprograms can be deleted from memory using the FREE statement.

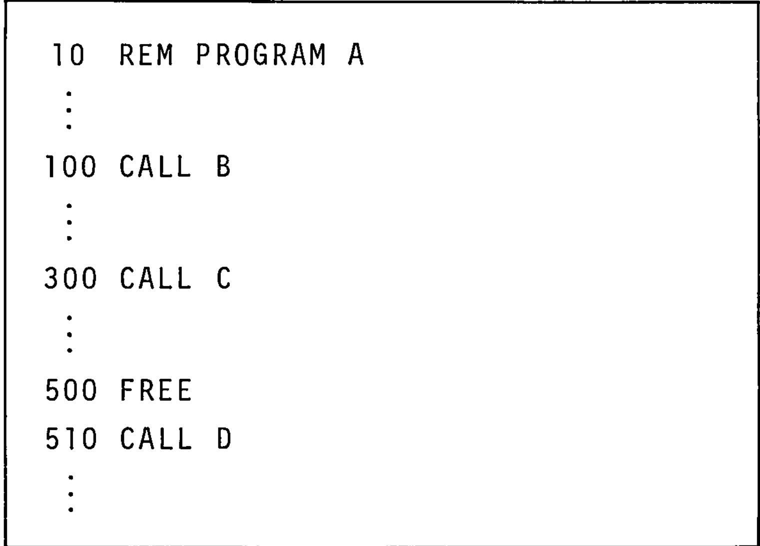


When the FREE statement is executed, all of the programs following the program in which the FREE statement is executed in memory, are deleted. When the FREE statement is executed in the main program, all of the subprograms called are deleted from memory. When the FREE statement is executed in a subprogram, all of the subprograms following the subprogram in memory are deleted from memory.



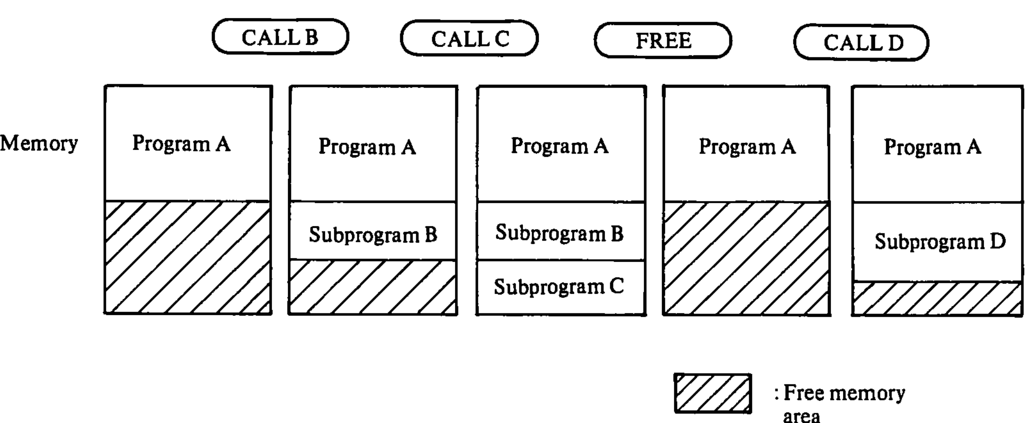
[Ex. 2.11.2-1]

Subprograms B and C are deleted from memory and then subprogram D is called.



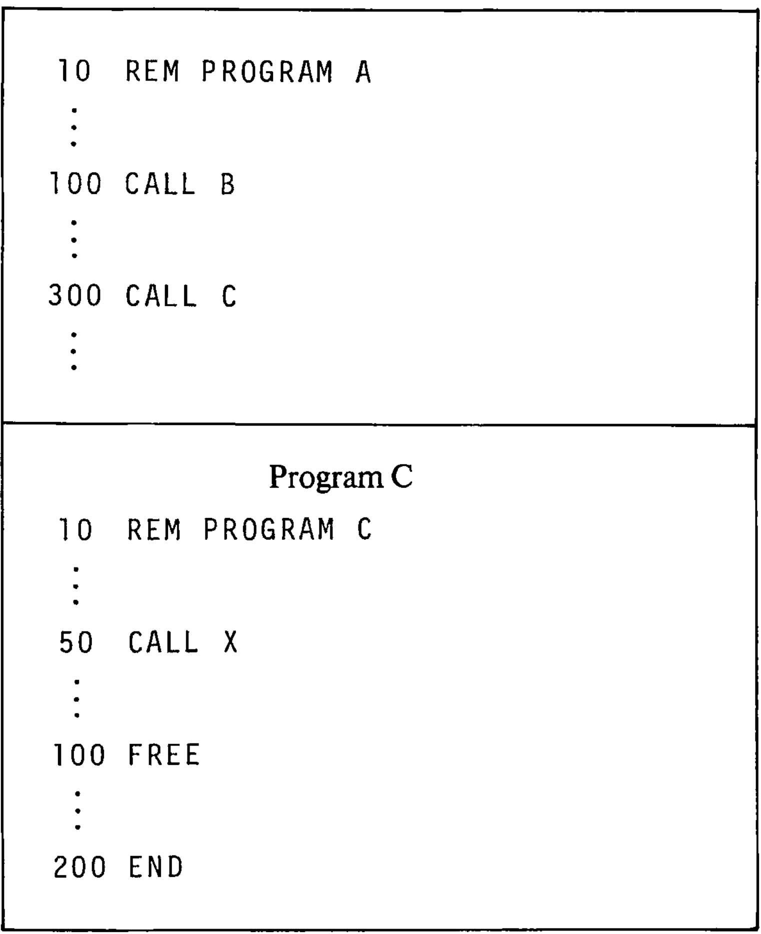
```
10  REM PROGRAM A
 :
100 CALL B
 :
300 CALL C
 :
500 FREE
510 CALL D
 :
```

Memory reservation is changed as follows:



[EX. 2.11.2-2]

In this example, the FREE instruction is executed in subprogram C.

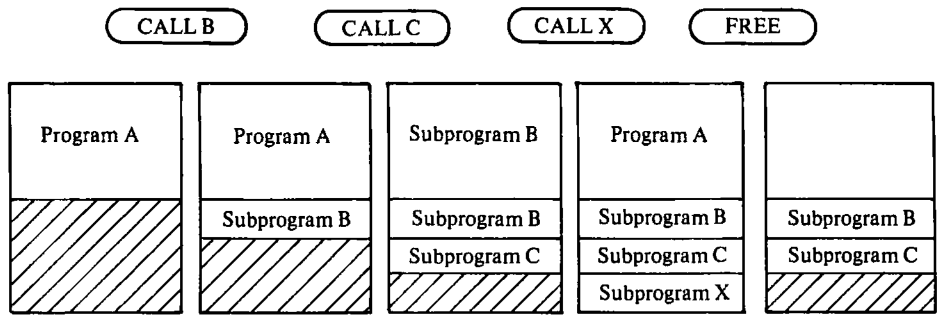


```
10  REM PROGRAM A
 :
100 CALL B
 :
300 CALL C
 :
```

Program C

```
10  REM PROGRAM C
 :
50  CALL X
 :
100 FREE
 :
200 END
```

Memory reservation is changed as follows:



In this case, only subprogram X is deleted from memory.

2.12 File-Related Instructions

2.12.1 OPEN Statement (Open)

Function

This instruction defines the logical device number for data files on disks or peripheral devices so they can be used in a program.

Format

```
OPEN␣#{<1~9>                 } ,{"[<Drive Name>]<File Name>"}
      {<Arithmetic Variable>}   {"<Device Name>"            }
                                {<String Expression>        }
```

Explanation

The OPEN statement must be executed before using data files on disks or peripheral devices.

The OPEN statement defines the logical device number for files or peripheral devices that will be used. The logical device number defined by the OPEN statement must be specified in instructions (e.g. PUT, GET, etc. described later) that perform input/output from/to the files and peripheral devices. Integers 1 ~ 9 can be specified as logical device numbers. If a logical device number is specified with an arithmetic variable, programs must be written to assign an integer 1 ~ 9 to the variable.

Specify a drive name when the file on disk is defined. When the drive name is omitted, the current drive is automatically specified.

Drive	Description on program
Floppy disk A	A: or FD0:
Floppy disk B	B: or FD1:
Floppy disk C	C: or FD2:
Floppy disk D	D: or FD3:
Mini floppy disk A	A: or FD0:
Mini floppy disk B	B: or FD1:

Refer to "1.10 Files" for the specification of file names. When the file type is omitted, DAT is automatically specified.

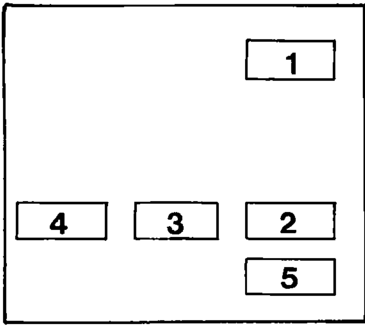
Specify the following device names when the peripheral devices are defined.

No.	I/O connector	Device name
0	Display unit or keyboard	CON: or CRT:
1	Centronics I/F	LPT: or UP0:
2	RS232C I/F (Centronics I/F)	US0: or TTY:(UL1: or UP1:)
3	RS232C (Centronics I/F)	PTR: or PTP:(UL1: or UP1:)
4	RS232C (Centronics I/F)	US1:(UL1: or UP1:)
5	RS232C (Centronics I/F)	US2:(UL1: or UP1:)

Note: Only one connector from connectors 2~5 can be used to connect a centronics I/F.

The numbers correspond to those in the figure below.

Rear View of Display Unit



The drive name, file name, or device name can also be specified with string expressions.

The definitions of the OPEN statement are valid until they are canceled by the CLOSE statement (described later). The definitions are automatically canceled when the program ends.

Only one file or peripheral device can be defined to a logical device number at a time. An error occurs if the OPEN statement is executed again for a file or peripheral device already defined by the OPEN statement.

[Ex. 2.12.1-1]
The disk file POINT is defined as logical device number 1.

```
 :
20   OPEN #1,"A:POINT"
 :
```

When POINT.DAT is on the disk in drive A, logical device number 1 is defined to it. When it is not, a new file POINT.DAT is created on the disk in drive A and defined as logical device number 1.

[Ex. 2.12.1-2]
A printer is connected to the Centronics I/F of connector 1 and defined as logical device number 2.

```
 :
20   OPEN #2,"LPT:"
 :
90   PRINT #2,"ABC"
 :
```

[Ex. 2.12.1-3]
Logical device numbers are defined according to the value (1~9) entered through the keyboard.

```
 :
90   INPUT A
100  OPEN #A,"LPT:"
 :
```

When 2 is entered through the keyboard on line 90, LPT: is defined as logical device number 2. If a numeric value other than 1 ~ 9 is entered on line 90, an error occurs on line 100.

[Ex. 2.12.1-4]

A disk file or peripheral device is defined by the character string entered through the keyboard.

```
 :
40   DIM A$10
 :
150  INPUT A$
160  OPEN #1,A$
 :
```

When "LPT:" is entered through the keyboard on line 150, LPT: is defined as logical device number 1. When "A: FILEGT" is entered, disk file FILEGT. DAT on the disk in drive A is defined as logical device number 1.

The DIM statement on line 40 specifies the number of characters that can be entered for the file name or device name as 10 characters (1 character for the drive name, 1 character for the colon, and 8 characters for the file name).

2.12.2 CLOSE Statement (Close)

Function

This instruction cancels the definition of a logical device number.

Format

```
CLOSE␣#{<1~9>                 } [ ,% DEL]
       {<Arithmetic Variable> }
```

Explanation

The contents of the logical device number defined by the OPEN statement are canceled (closed).

After the logical device number definitions are canceled (closed) by the CLOSE statement, they can be redefined.

Specify the logical device number that will be closed with 1~9. It also can be specified by arithmetic variable. When a logical device number is specified with an arithmetic variable, write the program so that the value of the variable is 1~9.

A file can be deleted from the disk after being closed by specifying % DEL (Delete). This specification is valid only when a disk file is closed.

[Ex. 2.12.2-1]

Logical device number #1 opened by the OPEN statement is closed.

```
 :
40   OPEN #1,"LPT:"
 :
170  CLOSE #1
 :
```

[Ex. 2.12.2-2]

Data file TRN.DAT on the disk on drive A is deleted at the same time logical device number #1 is closed. So if file TRN.DAT is opened after this, a new file is created on the disk.

```
 :
40   OPEN #1,"A:TRN"
 :
170  CLOSE #1,%DEL
 :
```

2.12.3 CHANGE Statement (Change)

Function

This instruction is used to change disks in a drive during the execution of a program.

Format

```
CHANGE␣[MSG (<String Expression>),]"<Drive Name> "
```

Explanation

In BASIC, mini floppy disks and floppy disks can usually be set into and removed from drives only when "$—" is displayed. If a disk is set or removed in another system status, the disk may be damaged or a data irregularity may occur in the file.

But when the CHANGE statement is executed in the program, a disk can be set/removed into/from the drive.

When the CHANGE statement is executed, the cursor is displayed and program execution is suspended temporarily. At this time, it is possible to set/remove a disk into/from the drive specified by the CHANGE statement. After disk replacement, depress ↵ to resume program execution.

When a message is specified after MSG in the operand of the CHANGE statement, this message is displayed with the cursor during temporary program suspension.

Before the CHANGE statement is executed to replace the disk, the disk file opened on the disk must be closed with the CLOSE statement.

[Ex. 2.12.3-1]

Processing is resumed after replacing of the disk in drive B.

```
 :
50   OPEN #1, "B:SALES1"
 :
90   CLOSE #1
100  CHANGE MSG("CHANGE THE DISK IN DRIVE B"),"B:"
110  OPEN #1, "B:SALES2"
 :
```

Operations after the execution of line 100 are shown next.

CHANGE

```
CHANGE THE DISK IN DRIVE B_
```

If ⏎ is depressed after disk replacement, program execution is resumed starting from line 110.

2.12.4 PUT Statement (Put)

Function

This instruction writes data to a file on a disk.

Format

```
PUT␣#{<1~9>              }[ ,<Arithmetic Expression>]␣<Variable>[ ,] ···
     {<Arithmetic Variable>}
```

Explanation

The PUT statement writes the contents of variables to a file on a disk. The unit for data writing is called a record. It is determined by the total length of the variables specified in the operand of the PUT statement. The lengths of variables are usually as follows:

Integer-type variable: 2 bytes
Real number-type variable: 8 bytes
String variable: 8 bytes

The length of a string variable can be changed to 1~255 bytes.

Each time the PUT statement is executed, one data record is written to a file.

There are two methods of writing data to a file, called sequential access and random access. Access means writing data to a file or reading data from the file.

In sequential access, file access is always performed starting from the first record of a file. In random access, the position of a record that will be accessed can be specified by record number.

The operating system reads/writes a data to a file on disk in 128-byte units. So, the end of the data written by the PUT statement isn't necessarily the same as the end of the file. (Details are described later.)

Therefore, the EOD (End of Data) record, which indicates the end of the data, must be written in the file at the end of the data written by use of the PUT statement. The EOD record is used only as the marker, so use a data which will not be confused with the file data.

The EOD record is effective when data is read by sequential access. When the contents of data (number of records and record length) are known, the EOD record isn't necessary.

Data writing to a file by the PUT statement is performed as follows:

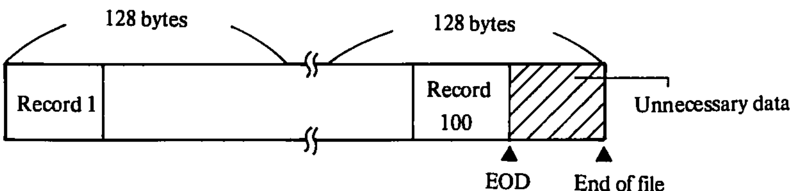
The CP/M-86 operating system writes data to files in 128-byte units. So, the size of the file can be calculated as follows:

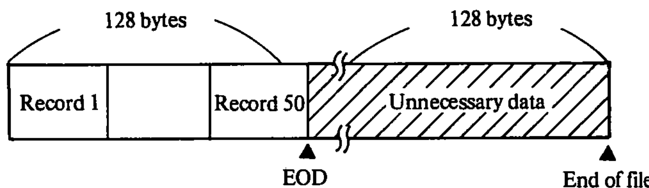
$$\left\lceil \frac{\text{Total record length}}{128} \right\rceil \times 128 \text{ bytes} = \text{File size}$$

Note: ⌊ ⌋ indicates that fractions are rounded up.

When ten 8-byte records are written to a file, the total record length is 80 bytes (=8 × 10) but the file size is 128 bytes.

Increasing the data volume in the file is automatically expands the file. But once a file is expanded, it cannot be made smaller. For example, even if a file is rewritten to include 100 data records and then the number of records is reduced to 50, the file size does not change.





Before executing the PUT statement, the file must be opened by the OPEN statement.

Note

The CP/M-86 operating system manages the disk in blocks, and consequently the file reserves the disk area in block units of 2K-bytes. So even a file whose size is only 128-byte reserves 2K-byte disk area.

Sequential Access

When a file is opened by the OPEN statement, the write starting position (indicated by what is called a pointer) is at the beginning of the file. Once the PUT statement is executed, the pointer moves to the next write position immediately following the first record. So in sequential access, records are written sequentially beginning at the head of the file.

For sequential access, <Arithmetic Expression> which specifies a record number (described later), is omitted.

OPEN #1 ....

File

Pointer →

PUT #1 (Record A)

(Record A)

Pointer →

PUT #2 (Record B)

(Record A)

(Record B)

Pointer →

PUT #1 (EOD Record)

(Record A)
(Record B)
(EOD Record)
Pointer →

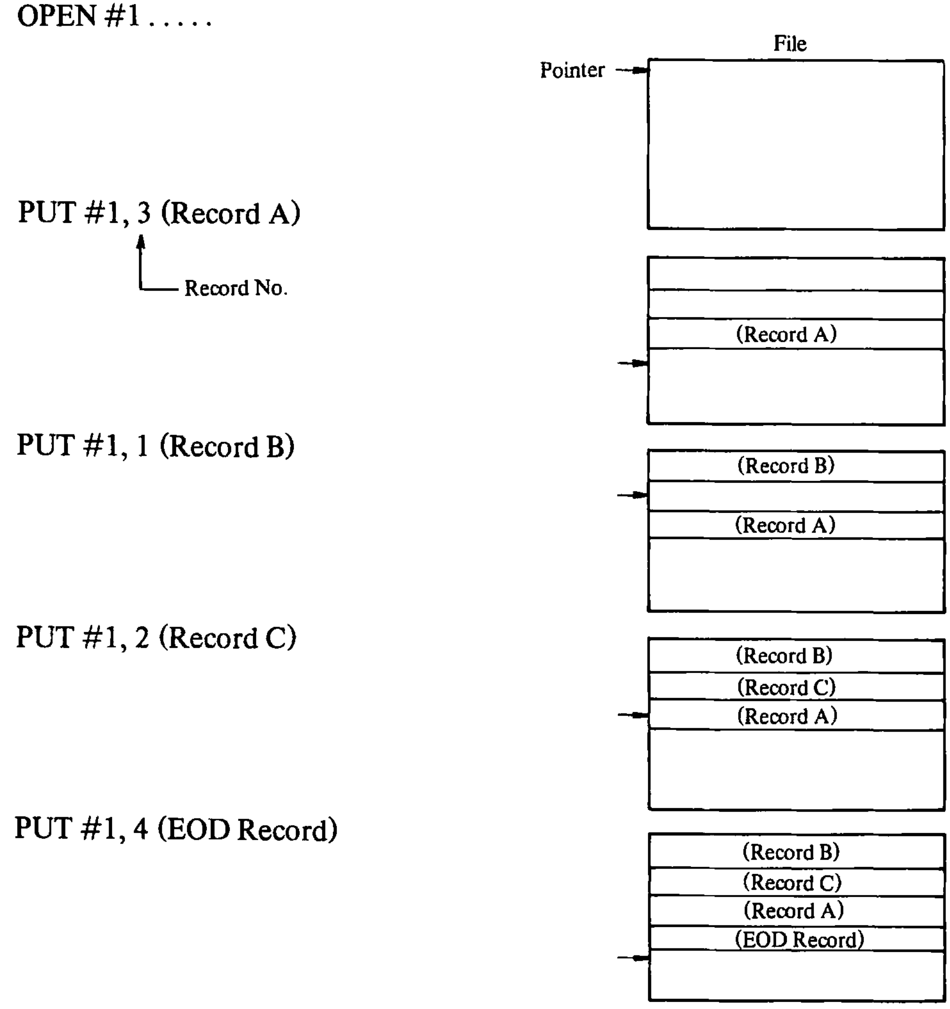
CLOSE #1 . . . . .

Random Access

In the execution of the PUT statement using random access, the pointer moves according to record numbers and records are written at the position indicated by pointer. After the data is written, the pointer is positioned immediately following the last record written.

For random access, record numbers must be specified in <Arithmetic Expression>.

0 cannot be specified as a record number.

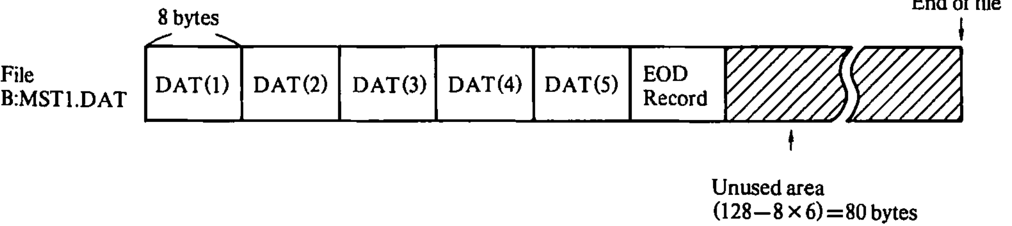


[Ex. 2.12.4-1]

The contents of variables DAT(1) ~ DAT(5) are written to a file by sequential access.

```
10   DIM DAT(5)
20   EODREC=9.9999999999999E63
 :
140  OPEN #1,"B:MST1"
150  FOR I=1 TO 5
 :
200  PUT #1 DAT(I)
 :
300  NEXT I
310  PUT #1 EODREC
320  CLOSE #1
```

In the above example, the PUT statement is executed five times in a loop on lines 150~300, and the contents of variables DAT(1) to DAT(5) are written by sequential access. 9.9999999999999E63 is written immediately following the data as an EOD record.

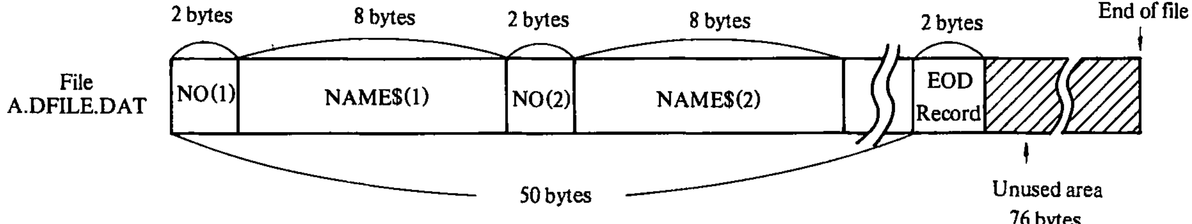


[Ex. 2.12.4-2]

Data blocks of different lengths are written to a file by sequential access.

```
10   INTEGER NO,EODREC
20   DIM NO(5),NAME$(5)
 :
200  OPEN #1,"A:DFILE"
210  FOR I=1 TO 5
 :
300  PUT #1 NO(I)
 :
400  PUT #1 NAME$(I)
 :
500  NEXT I
510  EODREC=32767
520  PUT #1 EODREC
530  CLOSE #1
```

In the above example, the record length in the PUT statement on line 300 is 2 bytes and the record length in the PUT statement on line 400 is 8 bytes. These two PUT statements are repeated five times in a loop on lines 210~500 by sequential access. Data blocks of different lengths are written as shown below.

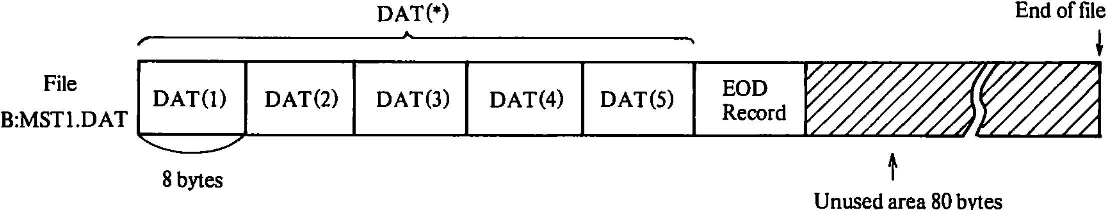


[Ex. 2.12.4-3]

The contents of array variables are written as one record to a file by specifying an asterisk (*).

```
10   DIM DAT(5)
20   EODREC=9.9999999999999E63
 :
100  OPEN #1,"B.MST1"
 :
200  PUT #1 DAT(*)
 :
290  PUT #1 EODREC
300  CLOSE #1
 :
```

The above example is a modified form of Ex. 2.12.4-1. Here an asterisk is used to specify all of the elements of array variables DAT(1)~DAT(5) as one record.

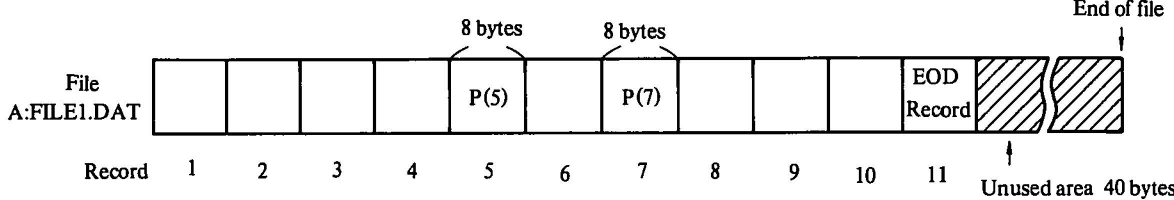


[EX. 1.12.4-4]
Data is written to a file by random access.

```
10   DIM P(10)
 :
100  OPEN #1,"A:FILE1"
 :
200  PUT #1,5 P(5)
 :
300  PUT #1,7 P(7)
 :
390  EODREC=9.9999999999999E63
400  PUT #1,11 EODREC
410  CLOSE #1
 :
```

In the above example, the data of variables P(5) and P(7) are written to a file by random access.

Both of the variables specified in the operands of the PUT statement on lines 200 and 300 have lengths of 8 bytes. Data is written to a file as shown below.



[Ex. 2.12.4-5]

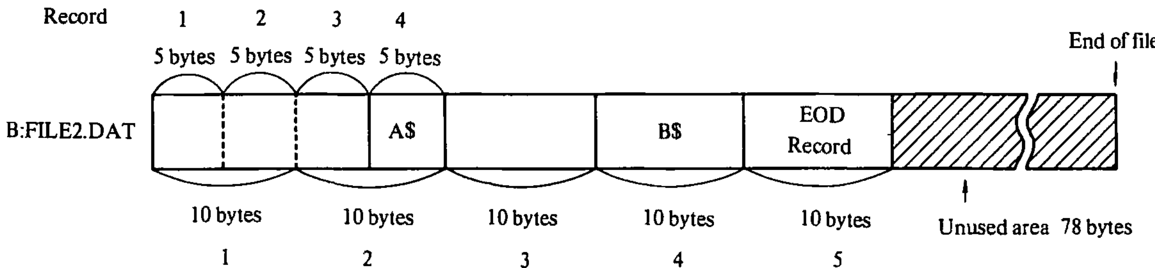
Data blocks of different lengths are written into a file by random access.

```
10   DIM A$10,B$5,EOD$10
20   EOD$="END-DATA"
 :
50   OPEN #1,"B:FILE2"
 :
100  PUT #1,4 A$
 :
200  PUT #1,4 B$
 :
300  PUT #1,5 EOD$
310  CLOSE #1
 :
```

The record length in random access is not fixed. The length of the variable specified in the operand of the PUT statement determines the record length. In the above example, because variable A$ in the PUT statement on line 100 has a length of 10 bytes, the contents of A$ are written beginning at the 31st byte from the head of the file (at the head of the 4th record in lengths of 10 bytes).

Likewise on line 200, writing starts at the 16th byte from the head of the file.

When data of different record lengths are written to a file by random access, be careful about the specification of record numbers.



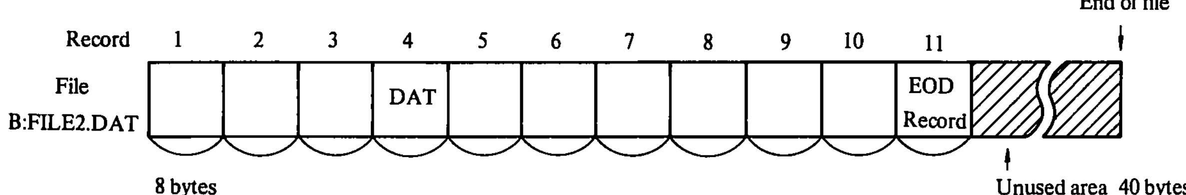
[Ex. 2.12.4-6]

Record numbers are entered through the keyboard and data is written to a file by random access.

```
100  OPEN #1,"B:FILE2"
 :
200  INPUT RNO
210  IF(RNO>10)OR(RNO<1) GOTO 200
220  PUT #1,RNO DAT
 :
290  EODREC=9.9999999999999E63
300  PUT #1,11 EODREC
310  CLOSE #1
```

On line 200, values 1~10 are entered to variable RNO as record numbers. If the input values exceed this range, line 200 is executed again by the IF statement on line 210. Data is written into the file by the PUT statement on line 220 according to the record number entered. If the RNO value is a fraction, the fractional part is automatically truncated.

When the input value is 4, the file is as shown below.



2.12.5 GET Statement (Get)

Function

This instruction reads data from a file on a disk.

Format:

```
GET␣#{<1~9>                  }[ , <Arithmetic Expression>]␣<Variable>[ , ] ···
     {<Arithmetic Variable>  }
```

Explanation

The GET statement reads data from a file on a disk and assigns the data to a variable. This statement is used for files in which data were written by the PUT statement. It is necessary to know which data were written in what order.

With the execution of the GET statement, one record of data is read from a file and assigned to variables. At this time, unless the type and length of data, etc, match that of the record data written, an incorrect value is assigned to variables.

Just as with the PUT statement, there are two kinds of file access, sequential access and random access. For details, refer to the PUT statement.

Data is read from files in units of 128 bytes by the operating system. So, except when the length of data written in a file is an integral multiple of 128 bytes, the end of data isn't the same as the end of the file. It is necessary, therefore, to determine the end of data using the EOD record described in the PUT Statement.

Like the INPUT statement, the GET statement has an automatic branch function based on no-input. In the GET statement, the no-input branch occurs when the data is read beyond the end of the file. At the same time, the value of the EOF function becomes −1.

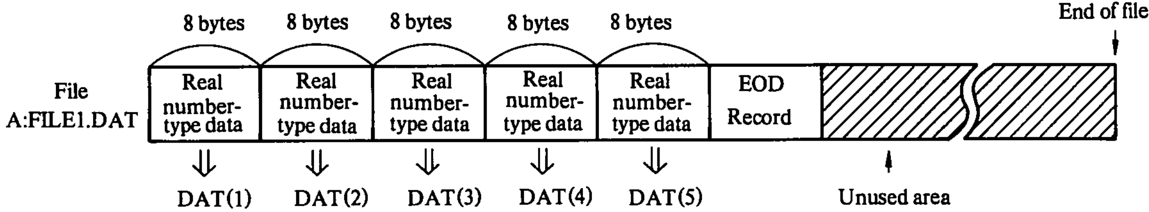
The file must be open by the OPEN statement before executing the GET statement.

[Ex. 2.12.5-1]

Data is read from a file by sequential access.

```
10   DIM DAT(5)
 :
100  OPEN #1,"A:FILE1"
110  FOR I=1 TO 5
120  GET #1 DAT(I)
130  NEXT I
 :
200  CLOSE #1
```

In this example, data is read sequentially from the top of data file A:FILE1.DAT in which five records of real number-type data (8-byte) is stored.



In the above example, the number of records written to the file is known, so it is not necessary to check the EOD record. But when the number of records written to a file is not known, the end of the data must be determined as follows:

```
10   DIM DAT(15)
 :
100  OPEN #1,"A:FILE1"
110  I=0
120  I=I+1
130  GET #1 DAT(I):IF DAT(I)<>9.9999999999999E63 GOTO 120
140
```

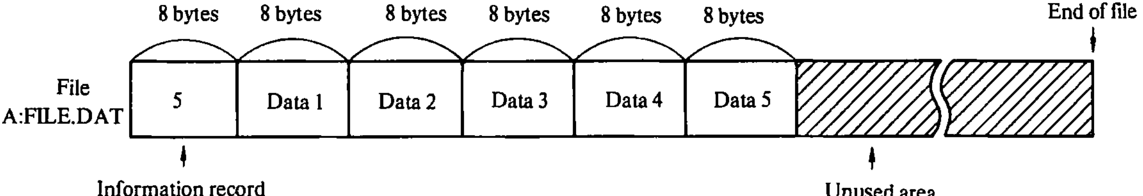
In the above example, the IF statement is used to check whether the data read on line 130 is the EOD record data. When the EOD record is read, reading ends and execution proceeds to line 140.

In the previous example, if data is read from a file that does not contain a EOD record or from a file in which the contents of the EOD record is other than 9.9999999999999E63, the no-input condition occurs at the end of the file and execution proceeds to line 140. At this time, the value of the EOF function becomes −1.

Advice

When data is read from a file by sequential access, an EOD record is required to determine the end of the data unless the number of records in the file is known. The EOD record can have any contents but some standardization is recommended to prevent confusion.

It is also better not to set any EOD record but to use the first record of a file as a file information record to indicate the number of records in a file.



The program shown below is used to read data from a file.

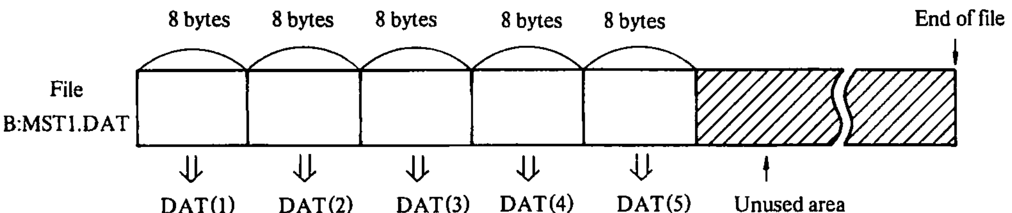
```
10   OPEN #1,"A:FILE"
20   GET #1 RNO
30   DIM DAT(RNO)
40   FOR I=1 TO RNO
50   GET #1 DAT(I)
60   NEXT I
70   CLOSE #1
```

[Ex.2.12.5-2]

Data are read to all elements of the array variables by specifying an asterisk (*).

```
10   DIM DAT(5)
 :
100  OPEN #1,"B:MST1"
 :
200  GET #1 DAT(*)
 :
300  CLOSE #1
 :
```

As in Ex. 2.12.4-3, the data are read and assigned to all elements ($8\times5=40$ bytes) of array variables DAT(1)~DAT(5) as one record at the same time.



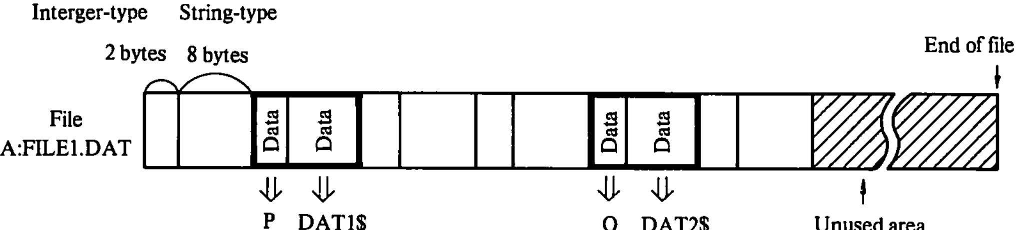
In this example, the data is read on the assumption that the number of records and data type in the file are known.

[Ex. 2.12.5-3]

Data is read from a file by random access.

```
10   INTEGER P,Q
 :
100  OPEN #1,"A:FILE1"
 :
200  GET #1,2 P,DAT1$
 :
300  GET #1,5 Q,DAT2$
 :
400  CLOSE #1
 :
```

In the above example, each record is 10 bytes (integer-type: 2 bytes, string type: 8 bytes) in the GET statements on line 200 and line 300. So, the data is read from the file as shown below. In this example, it is assumed that the contents of the file are known.



In this example, if the GET statement specifying a record number larger than 6 is executed, incorrect data is read. If reading is performed beyond the end of the file (i.e. record number greater than 12), no-input branch occurs and the value of the EOF function becomes −1.

[Ex. 2.12.5-4]

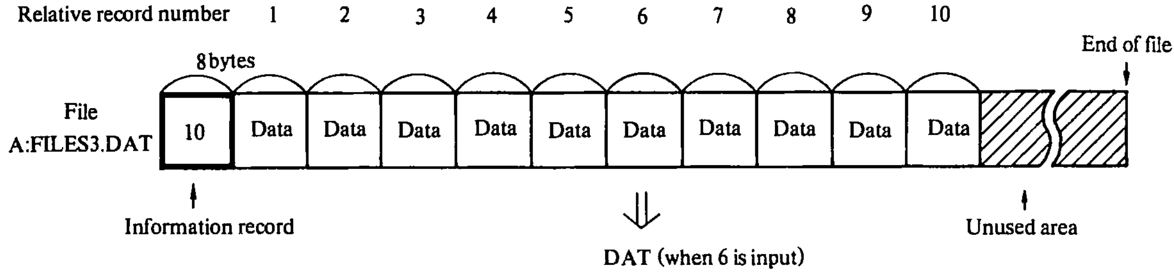
A record number is entered through the keyboard and data is read by random access:

```
  1   OPEN #1,"A:FILE3"
  2   GET #1 RNO
  :
100   INPUT X
110   IF (X<1)OR(X>RNO) GOTO 100
120   GET #1,X+1 DAT
  :
200   CLOSE #1
```

When a record number is entered through the keyboard, it is compared with the number written as the first record in a file indicating the number of records in the file. In random access, even if an EOD record is written at the end of data, it is not effective because the record read by random access is arbitrary.

In this case, it is useful to write an information record indicating the number of records as explained before.

In this example, the data in the file shown below is read by random access. If a value larger than the range of records is entered, it is checked by the IF statement on line 110 and line 100 is executed again. In the GET statement on line 120, 1 is added to the record number entered because record 1 is only an information record and is ignored. This means that data record 2 can actually be specified with relative record number 1.



[Ex. 2.12.5-5]

Data from an external input device is read by the GET statement.

```
10   DIM A$1
 :
100  OPEN #1,"US1:"
110  GET #1 A$
120  IF A$="&20" GOTO 210
 :
200  GOTO 110
210  CLOSE #1
 :
```

In the above example, the GET statement is used to read 1-byte data from an external input device connected to I/O connector 4 (US1:). The data is then assigned to 1-byte string variable A$. In this case, in the GET Statement on line 110, program execution is suspended until a 1-byte code is sent from the external input device. In the above example, execution of the data read routine ends when a space code (20_H) is sent as data.

2.12.6 Other Input and Output

When the logical device number defined by the OPEN statement described before is specified by I/O-related instructions (INPUT, INPUT USING, PRINT, and PRINT USING statements), input and output can be performed from/to files and external I/O devices using these instructions.

This section gives examples and explanations of input and output from/to files and external I/O devices.

(1) INPUT Statement

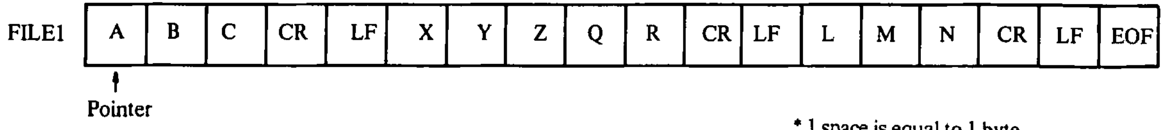
[Ex. 2.12.6-1]

Data is read from a file.

```
120  OPEN #1,"A:FILE1"
 :
200  INPUT #1,A$
 :
400  CLOSE #1
```

In the above example, data is read from file "FILET.DAT" by the INPUT statement. The reading of data from a file by the INPUT statement is performed in almost the same way as sequential access by the GET statement. Reading ends when the CR code ($0D_H$) and the LF code ($0A_H$) are read.

Assume that file "FILE1.DAT" has the following contents and the pointer is at the head of the file.

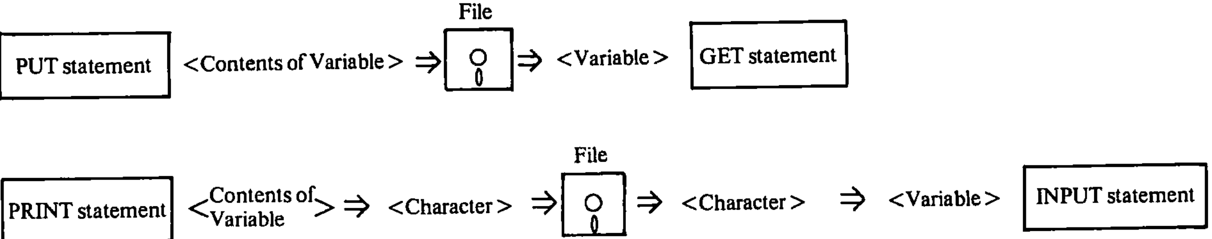


* 1 space is equal to 1 byte.

When the INPUT statement on line 200 is executed under these conditions, "ABC" is assigned to variable A$ and the pointer moves to position "X". After data reading is repeated, the pointer reaches position "EOF code ($1A_H$)", which indicates the end of the file. Then when the EOF code is read by the INPUT statement, the value of the EOF function (described later) becomes −1 and the no-input branch occurs, like when keyboard entry is performed. So data reading is not performed and execution proceeds to the next line.

As previously mentioned, the CR and LF codes are regarded as the end of one data when data is read from a file by the INPUT statement. So, a file read by the INPUT statement is usually created by the PRINT statement without a symbol at the end of statement. The EOF code ($1A_H$) is automatically written to files created by the PRINT or PRINT USING statements.

In input to and output from a file by the PRINT or INPUT statement, the correspondence between variables and data in the file do not have to be considered. These statements are different from the PUT and GET statements in this respect. Data from a character file created by the PRINT statement is regarded by the INPUT statement as a data input operation through a keyboard.



(2) INPUT USING Statement

Reading data from a file by the INPUT USING statement is almost the same as that for the INPUT statement, but the CR and LF codes are not necessary to indicate the end of one data.

[Ex. 2.12.6-2]
Data is read from a file.

```
100  OPEN #1,"A:FILE"
 :
200  INPUT #1,USING 210 A$
210  FORMAT ###
 :
400  CLOSE #1
```

In the above example, the contents of file "FILE" are as follows:

FILE	A	B	C	D	E	F	G	H	I	J	K	L	M	EOF

When the pointer is at the beginning of the file on line 200, the three characters specified by the FORMAT statement on line 210 are read from the beginning of the file and assigned to variable A$. That is, A$ = "ABC" and the pointer moves to positon "D", the next character, and execution proceeds to line 220.

At the end of the file, if there are fewer digits in the remaining data in a file than the number of digits specified to be read, the excess digits are automatically filled by NUL code (00_H).

So even when the pointer points at the EOF code, NUL codes equal to the number of input digits are assigned to the variable by the INPUT USING statement. And because the value of the EOF function becomes −1 when the pointer moves to the EOF positon, the end of file can be determined.

[Ex. 2.12.6-3]

Numeric data is read in.

```
150  OPEN #1,"A:FILE"
 :
200  INPUT #1,USING 210 A,B,C
210  FORMAT ##
 :
400  CLOSE #1
```

Assume that the file "FILE" is as follows:

FILE	1	2	3	4	5	6	7	8	9	0	EOF	

By executing the INPUT USING statement on line 200, two digits of data are read from the file and assigned to variables A, B, and C. So, the values of the variables are: A=12, B=34, and C=56. Then the pointer moves to "7".

Reading data from an external input device using the INPUT or INPUT USING statement is similar to reading data from a file. Remember that 1 byte of data (1 character) is regarded as one key operation. Be careful about the end of one data (CR and LF code) and the EOF code that indicates the end of the file.

(3) PRINT Statement

As explained before, 1-byte ASCII codes are output to a file or an external output device according to specification by the PRINT statement. But when data is output to a file using the PRINT statement, an EOF code ($1A_H$) is written to the end of the data as a data end code, so it is possible to detect the end of data when the data is read from the file by the INPUT or INPUT USING statements. In the GET statement, the EOF code is always regarded as a data.

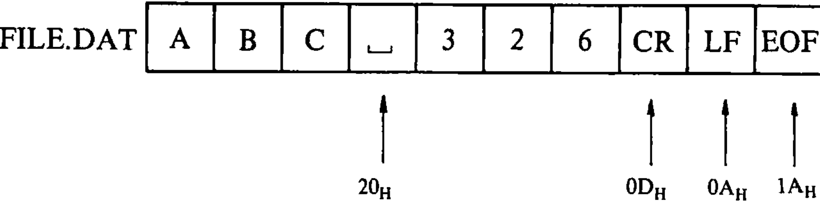
Output to an external output device like a printer is similar to output to a display unit. Depending on the specifications of printers, results may differ even if the same data are output. Load a handler before activating BASIC to use a printer.

The PRINT USING statement is used exactly like the PRINT statement.

[Ex. 2.12.6-4]
Output to a file

```
200  OPEN #1,"A:FILE"
 :
300  A$="ABC":X=326
310  PRINT #1,A$;X
 :
400  CLOSE #1
```

In the above example, the following file is created:



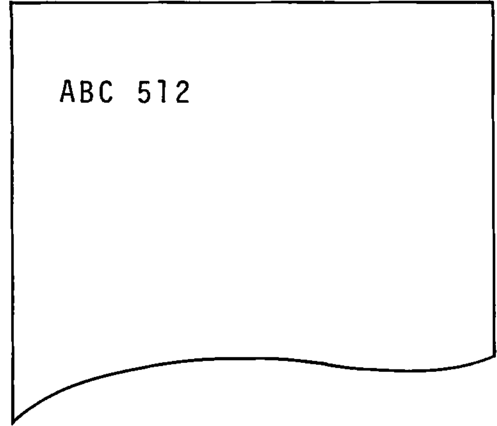
[Ex. 2.12.6-5]

Output to a printer.

```
200  OPEN #1,"LPT:"
 :
300  A$="ABC":X=512
310  PRINT #1,A$;X
 :
400  CLOSE #1
```

In the above example, data is output to a printer connected to I/O connector no.1. The printout is as follow:

Printout



2.13 Matrix-Related Instructions

2.13.1 Before Using Matrix Related Instructions

(1) Matrix

In a table, numeric values and characters are usually arranged in a rectangular or square format. The table below shows the performance of two pupils, X and Y, in Science, English, and Mathematics.

	Science	English	Mathematics
X	70	66	60
Y	60	75	80

This table can be represented as follows using parentheses:

$$\begin{pmatrix} 70 & 66 & 60 \\ 60 & 75 & 80 \end{pmatrix}$$

This type of figure arrangement is called a matrix. The individual figures are called elements of the matrix.

In the above example, a horizontal line of figures (e.g. 70, 66, and 60) is called a row. A vertical line of figures (e.g. 70 and 60) is called a column.

Generally, a matrix that has m rows and n columns is called an (m, n) matrix. A matrix that has equal numbers of rows and volumns is called a square matrix. An (n, n) matrix is called a square matrix of the nth degree.

A matrix is described as follows:

$$[A] = \begin{pmatrix} a_{11} & a_{12} \cdots a_{1n} \\ a_{21} & a_{22} \cdots a_{2n} \\ & \vdots \\ a_{m1} & a_{m2} \cdot\cdot\ a_{mn} \end{pmatrix}$$

(2) Unit Matrix

A unit matrix is a square matrix in which all of the elements on the diagonal line from left top to right bottom are 1 and the other elements are all 0. The unit matrix is represented by I.

In matrix calculations, the unit matrix corresponds to 1 in numeric calculations.

Ex.

$$[I] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(3) Transposed Matrix

Whe matrix [A] consists of i rows and j columns, the matrix formed by replacing the i rows by the j rows and the j columns by the i columns is called a transposed matrix, which is represented by $[A]^t$.

Ex.

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & 1 & -1 \end{bmatrix}^t = \begin{bmatrix} 1 & 0 \\ 3 & 1 \\ 2 & -1 \end{bmatrix}$$

(4) Inverse Matrix

In a calculation with real numbers, the value of b that satisfies $ab=1$ or $ba=1$ when $a \neq 0$ is called the inverse of a. Likewise, when the product of two square matrixes is equal to unit matix [I] (i.e. [A][B] = [I]), [B] is called the inverse matrix of [A] and [A] is the inverse matrix of [B]. They are represented by $[A]^{-1}$ and $[B]^{-1}$ respectively.

At this time, the following expression is valid:

$$[A][A]^{-1} = [A]^{-1}[A] = [I]$$

(5) **Sum and Difference of Matrixes**

When matrixes [A] and [B] have an equal number of rows and cloumns, the matrix formed with the sums or differences of the corresponding elements of the two matrixes is called the sum or difference of [A] and [B]. It is represented by [A] + [B] or [A] − [B].

Ex.

$$[A] = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, [B] = \begin{pmatrix} 4 & 5 & 6 \\ 1 & 3 & 2 \end{pmatrix}$$

$$[A] + [B] = \begin{pmatrix} 5 & 7 & 9 \\ 5 & 8 & 8 \end{pmatrix}$$

$$[A] - [B] = \begin{pmatrix} -3 & -3 & -3 \\ 3 & 2 & 4 \end{pmatrix}$$

(6) **Product of Matrixes**

The product of matrixes is calculated as follows:
Take as a simple example:

$$[A] = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, [X] = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

The product of [A] [X] is:

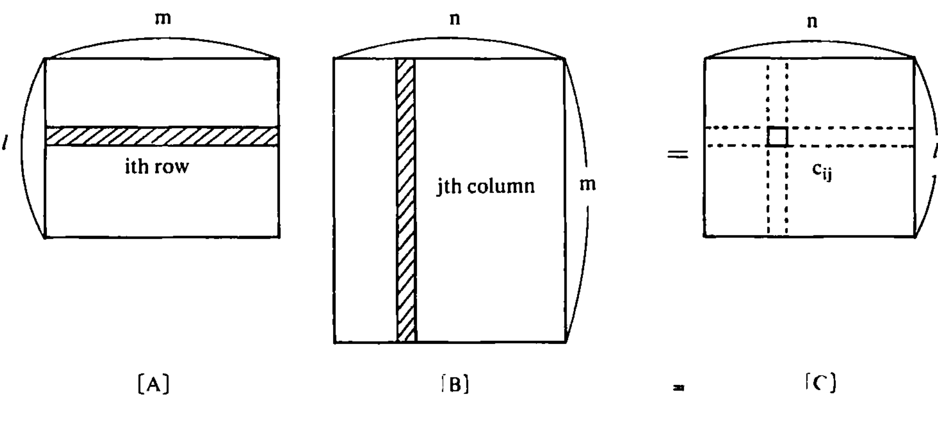
$$[A][X] = \begin{pmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \end{pmatrix}$$

The product of two square matrixes is:

$$[A] = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, [X] = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix}$$

$$[A][X] = \begin{pmatrix} ax_1 + bx_2 & ay_1 + by_2 \\ cx_1 + dx_2 & cy_1 + dy_2 \end{pmatrix}$$

The product of matrixes is represented as follows:



$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} = \cdots + a_{im}b_{mj} = \sum_{p=1}^{m} a_{ip}b_{pj}$$

$$\begin{pmatrix} i = 1, \cdots, l \\ j = 1, \cdots, n \end{pmatrix}$$

(7) Values of Determinants

A determinant is an expression in which the elements of square matrixes are calculated according to a certain rule. A determinant is represented by $|A|$. The result of the calculation is called the value of the determinant.

Take as an example a square matrix:

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix}$$

The value of the following expression is:

$$a_1b_2 - a_2b_1$$

is represented by:

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

This is called a determinant of the second degree. The value of $a_1b_2 - a_2b_1$ is called the value of the determinant.

A determinant of the third degree and its value are:

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = a_1(b_2c_3 - b_3c_2) - a_2(b_1c_3 - b_3c_1) + a_3(b_1c_2 - b_2c_1)$$

(8) Expansion of Determinants

To formulate a calculation expression from the elements of a determinant is called the expansion of the determinant.

- Second degree

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1b_2 - a_2b_1$$

- Third degree

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$$

$$= a_1(b_2c_3 - b_3c_2) - a_2(b_1c_3 - b_3c_1) + a_3(b_1c_2 - b_2c_1)$$

- Fourth degree

$$\begin{vmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \\ b_4 & c_4 & d_4 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & c_1 & d_1 \\ b_3 & c_3 & d_3 \\ b_4 & c_4 & d_4 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_4 & c_4 & d_4 \end{vmatrix} - a_4 \begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \end{vmatrix}$$

Generally, to expand a determinant of the nth degree, minus signs are given to the even elements in the first column. Then the sub-determinant of the elements in the first column is obtained. A sub-determinant is a determinant of the $(n-1)$th degree determined by removing the row and column that contain the element from a given determinant finally the aggregate sum of the products of the elements in the first column with the signs and the sub-determinants of the elements is obtained.

(9) Characteristics of Determinants

Determinants have the following characteristics:

- Even if the rows and the columns are switched, the value of the determinant does not change.

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$$

- If two adjacent rows or columns are exchanged, the sign of the value of the determinant changes.
- The value of a determinant in which each element consists of the same rows or columns is 0.
- When the elements in a row are equal to m times the corresponding elements in another row, the value of the determinant is 0.

- If a row of a determinant is multiplied by *m*, the value of the new determinant is *m* times the value of the original determinant.
- $$\begin{vmatrix} a_1+p_1+q_1 \\ a_2+p_2+q_2 \\ a_3+p_3+q_3 \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} + \begin{vmatrix} p_1 & b_1 & c_1 \\ p_2 & b_2 & c_2 \\ p_3 & b_3 & c_3 \end{vmatrix} + \begin{vmatrix} q_1 & b_1 & c_1 \\ q_2 & b_2 & c_2 \\ q_3 & b_3 & c_3 \end{vmatrix}$$
- $$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = \begin{vmatrix} a_1+mb_1 & b_1 & c_1 \\ a_2+mb_2 & b_2 & c_2 \\ a_3+mb_3 & b_3 & c_3 \end{vmatrix}$$

(10) Solution of Simultaneous Linear Equations

As an application of matrixes, simultaneous linear equations are solved by the two different methods shown here.

• Method Using the Values of Determinants

When:

$$\begin{cases} ax+by=p \\ cx+dy=q \end{cases}$$

the simultaneous linear equation becomes:

$[A][X]=[D]$

where:

$$[A]=\begin{pmatrix} a & b \\ c & d \end{pmatrix},\ [X]=\begin{pmatrix} x \\ y \end{pmatrix},\ [D]=\begin{pmatrix} p \\ q \end{pmatrix}$$

When $ab-bc\neq 0$, the solution can be found by obtaining $[A]^{-1}$, the inverse matrix of $[A]$:

$[X]=[A]^{-1}[D]$

- Method of Using the Values of Determinants

When:

$$\begin{cases} a_1x + b_1y + c_1z = d_1 \\ a_2x + b_2y + c_2z = d_2 \\ a_3x + b_3y + c_3z = d_3 \end{cases}$$

assume:

$$D=\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \neq 0,\ D_1=\begin{vmatrix} b_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix},\ D_2=\begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix},\ D_3=\begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & d_3 & d_3 \end{vmatrix}$$

Then the solutions x, y, and z are obtained as follows:

$$x=\frac{D_1}{D},\ y=\frac{D_2}{D},\ z=\frac{D_3}{D}$$

2.13.2 Notes for Using Matrix-Related Instructions

Regarding integer-type and real number-type array variables of two dimensions as matrixes, matrix-related instructions perform various calculations. Therefore, array variables for a matrix must be defined by the DIM statement before executing matrix-related instructions. The method for definition is described below.

For example, in order to define array variable A as a matrix of 3 rows and 4 columns, specify:

DIM A(3, 4)

The subscript must be within the range: 1(0) ~ 32767.

Use an asterisk (*) to specify all elements of array variables for a matrix.

$$A(*) = \begin{bmatrix} A(1,1) & A(1,2) & A(1,3) & A(1,4) \\ A(2,1) & A(2,2) & A(2,3) & A(2,4) \\ A(3,1) & A(3,2) & A(3,3) & A(3,4) \end{bmatrix}$$

Each element of the matrix corresponds to each of the array variables.

The execution of a matrix-related instruction requires the loading of a matrix library. The library is loaded automatically when a matrix-related instruction is executed. But it is also possible for the library to reside in memory. Ehen the matrix library is resident in memory, it need not be loaded during program execution, so higher speed processing can be performed. For the specification of a resident matrix library, refer to the explanation of the BASIC commands in "Chapter II, 2.4.13 OS Mode Commands".

2.13.3 MAT INPUT Statement (Matrix Input)

Function

This statement assigns data to a matrix through the keyboard or a file.

Format

```
MAT INPUT(<Variable>, <Array Variable>(*))
```

Explanation

To enter data through the keyboard, specify 0 in <Variable> indicating the logical device number. The OPEN statement is not necessary to enter data through the keyboard.

When entering data from a file, specify the logical device number defined by the OPEN statement in <Variable>. The symbol # immediately preceding the logical device number must be omitted.

[Ex. 2.13.3-1]

Data is entered through the keyboard to matrix A, consists of 2 columns and 2 rows.

```
10   DIM A(2,2)
 :
100  MAT INPUT(0,A(*))
 :
```

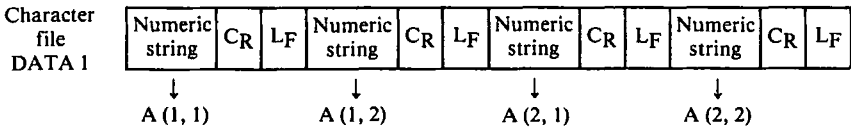
In this example, the function is exactly the same as the one in the following example using the INPUT statement.

```
10   DIM A(2,2)
 :
100  INPUT A(1,1),A(1,2),A(2,1),A(2,2)
 :
```

[Ex. 2.13.3-2]

As in the case of the INPUT statement, data is read into the matrix A from a character file on a disk.

```
10   DIM B(2,2)
50   OPEN #1,"B:DATA1"
 :
100  MAT INPUT(1,A(*))
 :
```



Data reading from a file starts at the pointer position just like in the INPUT statement. Each data in a character file must also have a CR code and LF code as the end of one data.

2.13.4 MAT READ Statement (Matrix Read)

Function

This statement assigns data specified by the DATA statement to a matrix.

Format

```
MAT READ(<Array Variable>(*))
```

Explanation

Data specified by a DATA statement are assigned to the array variable elements. The order of input is as follows:

For example, when A (1, 1) ~ A (2, 3), the order of input is A (1, 1), A (1, 2), A (1, 3), A (2, 1), A (2, 2) and A (2, 3). Follow the order of data specified by the DATA statement.

The RESTORE statement can also be used.

[Ex. 2.13.4-1]

Data is read to a matrix consisting of 2 rows and 3 columns.

```
10   DIM A(2,3)
 :
50   MAT READ(A(*))
 :
100  DATA 100,148,162,172,720,310
 :
```

In this example, the function is exactly the same as the one in the following example using the READ statement.

```
10   DIM A(2,3)
 :
50   READ A(1,1),A(1,2),A(1,3),A(2,1),A(2,2),A(2,3)
 :
100  DATA 100,148,162,172,720,310
 :
```

2.13.5 MAT PRINT Statement (Matrix Print)

Function

The MAT PRINT statement outputs the elements of a matrix.

Format

```
MAT PRINT(<Variable>, <Array Variable>(*))
```

Explanation

This statement sequentially outputs the elements of a matrix to a file or a device specified in <Variable> indicating the logical derice number.

Specify 0 for output to the display. The OPEN statement must also be excuted. The elements are displayed one after another in a horizontal line. For output in matrix form, use the PRINT statement or the PRINT USING statement.

For output to a file on a disk, specify the logical device number defined by the OPEN statement. The symbol # immediately preceding the logical device number must be omitted.

[Ex. 2.13.5-1]

The elements of matrix A consisting of 2 rows and 3 columns are displayed.

```
10   DIM A(2,3)
 :
50   MAT PRINT(0,A(*))
 :
```

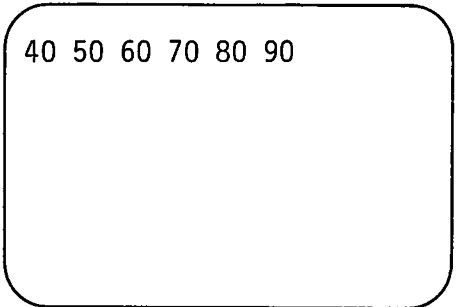
In this example, the function is exactly the same as the one in the following example using the PRINT statement.

```
10   DIM A(2,3)
 :
50   PRINT A(1,1);A(1,2);A(1,3);A(2,1);A(2,2);A(2,3)
 :
```

When the values of the variable are:

A (1, 1) = 40, A (1, 2) = 50, A (1, 3) = 60
A (2, 1) = 70, A (2, 2) = 80, A (2, 3) = 90

The following is output:

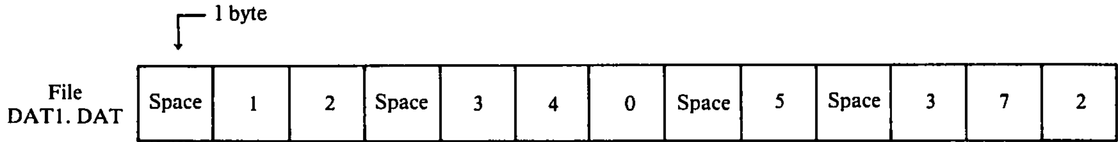


[Ex. 2.13.5-2]

The values of the elements of a matrix are written to a file on disk.

```
10   DIM A(2,2)
50   OPEN #1,"B:DATA1"
 :
100  MAT PRINT(1,A(*))
 :
```

In the above example, if the values of the array variables are: A (1, 1) = 12, A (1, 2) = 340, A (2, 1) = 5, and A (2, 2) = 372, the following character data are written to the file;



Note

When data is written to a file using the MAT PRINT statement, a CR, LF, or EOF code is not written at the end of the data.

2.13.6 MAT MOV Statement (Matrix Move)

Function

This statement assigns the values of the elements of a matrix to those of another matrix.

Format

```
MAT MOV(<Array Variable 2>(*), { <Array Variable 2>(*)   } )
                               { <Arithmetic Expression> }
```

Explanation

This instruction corresponds to the LET statement, which handles numeric values.

The value of each element of <Array Variable 1> is assigned to the corresponding element of <Array Variable 2>. In this case, the two matrixed must have an equal numbers of rows and columns.

If <Arithmetic Expression> is specified instead of <Array Variable 2>, the value of the specified expression is assigned to all elements of <Array Variable 1>.

[Ex. 2.13.6-1]

The element values of matrix B are assigned to those of matrix A.

```
10   DIM A(2,2),B(2,2)
 :
50   MAT MOV(A(*),B(*))
 :
```

$$A(*) = \begin{pmatrix} A(1,1) & A(1,2) \\ A(2,1) & A(2,2) \end{pmatrix} \quad B(*) = \begin{pmatrix} B(1,1) & B(1,2) \\ B(2,1) & B(2,2) \end{pmatrix}$$

[Ex. 2.13.6-2]

The 5 is assigned to each element of matrix A.

```
10   DIM A(2,2)
 :
50   MAT MOV(A(*),5)
 :
```

[Ex. 2.13.6-3]

The value of an arithmetic expression is assigned to each of the elements of matrix A.

```
10   DIM A(3,3)
 :
50   MAT MOV (A(*),X+Y)
 :
```

If X = 40 and Y = 73, each element of matrix A is 113.

2.13.7 MAT ADD Statement (Matrix Addition)

Function

This statement adds the values of the elements of a matrix to those of another matrix.

Format

```
MAT ADD(<Array Variable 1>(*), { <Array Variable 2>(*)  } )
                               { <Arithmetic Expression> }
```

Explanation

The value of each element of <Array Variable 2> are added to those of <Array Variable 1>. The two matrixes must have an equal numbers of rows and columns.

If <Arithmetic Expression> is specified instead of <Array Variable 2>, the value of the specified expression is added to the value of each element of <Array Variable 1>.

[Ex. 2.13.7-1]

The element values of matrix B are added to those of matrix A.

```
10   DIM A(2,2),B(2,2)
 :
50   MAT ADD(A(*),B(*))
 :
```

Assuming that: $[A] = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, [B] = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$

then on line 50;

$$[A] = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

[Ex. 2.13.7-2]

The value of variable X is added to each of the elements of matrix A.

```
10   DIM A(3,3)
 :
50   MAT ADD(A(*),X)
 :
```

When the value of variable X is 246, 246 is added to each of the elements of A (*).

2.13.8 MAT SUB Statement (Matrix Subtraction)

Function

The values of the elements of a matrix are subtracted from those of another matrix.

Format

```
MAT SUB(<Array Variable 1>, { <Array Variable 2> (*)  } )
                            { <Arithmetic Expression> }
```

Explanation

The values of the elements of <Array Variable 2> are subtracted from those of <Array Variable 1>. The two matrixed must have an equal number of rows and columns.

When <Arithmetic Expression> is specified instead of <Array Variable 2>, the value of the specified expression is subtracted from the value of each element of <Array Variable 1>.

[Ex. 2.13.8-1]

The element values of matrix B are subtracted from those of martix A.
A.

```
10   DIM A(2,2),B(2,2)
:
50   MAT SUB(A(*),B(*))
:
```

Assuming that: $[A]=\begin{pmatrix}10 & 9\\ 8 & 7\end{pmatrix}$, $[B]=\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$

then on line 50: $[A]=\begin{pmatrix}9 & 7\\ 5 & 3\end{pmatrix}$

[Ex. 2.13.8-1]

The value of an arithmetic expression is subtracted from each of the elements of matrix A.

```
10   DIM A(2,2)
:
50   MAT SUB(A(*),X+Y)
:
```

When X=20 and Y=30, 50 is subtracted from each of the elements of matrix A.

2.13.9 MAT MUL Statement (Matrix Multiplication)

Function

This statement multiplies the values of the elements of a matrix by those of another matrix.

Format

MAT MUL(<Array Variable 1>(*), { <Array Variable 2>(*) | <Arithmetic Expression> })

Explanation

The values of the elements of <Array Variable 1> are multiplied by the corresponding elements of <Array Variable 2>. The two matrixes must have an equal number of rows and columns.

Use the MAT MLT statement (described later) to obtain the product of the matrixes.

When <Arithmetic Expression> is specified instead of <Array Variable 2>, the value of each element of <Array Variable 1> is multiplied by the value of the specified expression

[Ex. 2.13.9-1]

The element values of matrix A are multiplied by those of matrix B.

```
10   DIM A(2,2),B(2,2)
:
50   MAT MUL(A(*),B(*))
:
```

Assuming that: $[A] = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, [B] = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$

then on line 50: $[A] = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$

[Ex. 2.13.9-2]

Each element of matrix A is multiplied by the value of an arithmetic expression.

```
10   DIM A(3,3)
:
50   MAT MUL(A(*),L*M)
:
```

When variable L = 7 and M = 8, each element of matrix A is multiplied by 56 (= 7 × 8).

2.13.10 MAT DIV Statement (Matrix Division)

Function

This statement divides the values of the elements of a matrix by those of another matrix.

Format

```
MAT DIV(<Array Variable 1>(*), { <Array Variable 2>(*)    } )
                                { <Arithmetic Expression> }
```

Explanation

The values of the elements of <Array Variable 1> are divided by those of <Array Variable 2>. The two matrixes must have an equal numbef of rows and columns.

When <Arithmetic Cxpression> is specified instead of <Array Variable 2>, the value of each element of <Array Variable 1> is divided by the value of the specified expression.

The elements of the array variables and the value of the arthmetic expression cannot be 0.

[Ex. 2.13.10-1]

The element values of matrix A are divided by those of matrix B.

```
10   DIM A(2,2),B(2,2)
:
50   MAT DIV(A(*),B(*))
:
```

Assuming that: $[A] = \begin{pmatrix} 10 & 15 \\ 18 & 20 \end{pmatrix}$, $[B] = \begin{pmatrix} 5 & 3 \\ 9 & 4 \end{pmatrix}$

then on line 50; $[A] = \begin{pmatrix} 2 & 5 \\ 2 & 5 \end{pmatrix}$

[Ex. 2.13.10-2]

Each element of matrix A is divided by the value of a variable.

```
10   DIM A(3,3)
 :
50   MAT DIV(A(*),S)
 :
```

When S = 100, each element of A (*) is divided by 100.

2.13.11 MAT SUM Statement (Matriex Sum)

Function

The sum of the values of the elements of a matrix is assigned to a variable.

Format

```
MAT SUM(<Variable>, <Array Variable> (*))
```

Explanation

The values of the elements of an array variable are totaled and the result is assigned to another variable.

[Ex. 2.13.11-1]

The element values of matrix B are totaled and the result is assigned to variable X.

```
10   DIM B(2,2)
 :
50   MAT SUM(X,B(*))
 :
```

Assuming that: $[B] = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

then on line 50: $X = 1 + 2 + 3 + 4 = 10$

2.13.12 MAT CSUM Statement (Matrix Column Sum)

Function

The total of the values of the elements in each column of a matrix is assigned to another matrix.

Format

```
MAT CSUM(<Array Variable>(*),<Array Variable 2>(*))
```

Explanation

The total value of the elements in each column of <Array Variable 2> is calculated and the result is assigned to <Array Variable 1>. The two matrixes must have an equal number of columns. <Array Variable 1> must also have only one raw.

Note

<Array Variable 1> must be one-dimensional and <Array Variable 2> must be two-dimensional.

[Ex. 2.13.12-1]

The values of the elements in each column of matrix B are totaled and the result is assigned to matrix A.

```
10   DIM A(2),B(3,2)
 :
50   MAT CSUM(A(*),B(*))
 :
```

Assuming that: $[B] = \begin{pmatrix} 4 & 5 \\ 7 & 8 \\ 10 & 11 \end{pmatrix}$

then on line 50: $[A] = (21 \quad 24)$

2.13.13 MAT RSUM Statement (Matrix Row Sum)

Function

The total of the values of the elements in each row of a matrix is assigned to another matrix.

Format

```
MAT RSUM(<Array Variable 1>(*),<Array Variable 2>(*))
```

Explanation

The total value of the elements in each row of <Array Variable 2> is calculated and the result is assigned to <Array Variable 1>. The two matrixes must have an equal number of rows. <Array Variable 1> must have only one column.

Note

<Array Variable 1> must be one-dimensional, and <Array Variable 2> must be two-dimensional.

```
10   DIM A(2),B(2,2)
 :
50   MAT RSUM(A(*),B(*))
 :
```

Assuming that: $[B] = \begin{pmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

then on line 50: $[A] = \begin{pmatrix} 15 \\ 24 \end{pmatrix}$

2.13.14 MAT IDN Statement (Matrix Identity)

Function

This statement constructs a unit matrix of the specified size.

Format

```
MAT IDN(<Array Variable>(*))
```

Explanation

A unit matrix is constructed by assigning 0 and 1 to the elements of an array variable of the size defined by the DIM statement. The array variable must be a square matrix.

[Ex. 2.13.14-1]

A unit matrix is constructed by assigning 0 and 1 to the elements of square matrix A.

```
10   DIM A(4,4)
 :
50   MAT IDN (A(*))
 :
```

On line 50:

$$[A]=\begin{pmatrix}1&0&0&0\\0&1&0&1\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

2.13.15 MAT INV Statement (Matrix Inverse)

Function

This statement constructs the inverse matrix of a specified square matrix.

Format

```
MAT INV(<Array Variable>(*))
```

Explanation

This statement calculates the inverse matrix of a matrix specified by the array variable and assignes the result to the original array. The array variable must be a square matrix. An error occurs if the specified matrix is not a square matrix.

Note

The array variable must be two-dimensional.

[Ex. 2.13.15-1]

The inverse matrix of matrix A is calculated.

```
10   DIM A(2,2)
 :
50   MAT INV(A(*))
 :
```

Assuming that: $[A] = \begin{pmatrix} 2 & 4 \\ 1 & 3 \end{pmatrix}$

then on line 50: $[A] = \begin{pmatrix} 1.5 & -2 \\ -0.5 & 1 \end{pmatrix}$ because $[A][A]^{-1} = [I]$.

2.13.16 MAT TRN Statement (Matrix Transpose)

Function

This statement constructs the transposed matrix of a specified matrix.

Format

```
MAT TRN(<Array Variable 1>(*),<Array Variable 2>(*))
```

Explanation

The transposed matrix of the matrix specified by <Array Variable 2> is assigned to the matrix specified by <Array Variable 1>. That is, the rows become colums and the columns become rows.

The number of the rows of <Array Variable 1> must be equal to the number of columns of <Array Variable 2>.

The number of the columns of <Array Variable 1> must be equal to that of the rows of <Array Variable 2>.

[Ex. 2.13.16-1]

The transposed matrix of matrix B is assigned to A.

```
10   DIM A(2,3),B(3,2)
:
50   MAT TRN(A(*),B(*))
:
```

Assuming that: $[B]=\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$

then on line 50: $[A]=\begin{pmatrix}1 & 3 & 5\\ 2 & 4 & 6\end{pmatrix}$

2.13.17 MAT DET Statement (Matrix Determinant)

Function

This statement calculates the value of the determinant of a specified square matrix and constructs the its inverse matrix.

Format

```
MAT DET(<Variable>, <Array Variable> (*))
```

Explanation

The determinant of a square matrix specified by <Array Variable> is calculated and the result is assigned to the variable. Simultaneously, the inverse matrix of the square matrix specified by <Array Variable> is assigned to the original matrix. When the determinant is 0, a meaningless value is assigned to the array variable because there is no inverse matrix.

[Ex. 2.13.17-1]

The value of the determinant of matrix A is calculated.

```
10   DIM A(2,2)
 :
50   MAT DET(X,A(*))
 :
```

Assuming that: $[A] = \begin{pmatrix} 2 & 4 \\ 1 & 3 \end{pmatrix}$

then on line 50: $X = 2 \times 3 - 4 \times 1 = 2$

$$[A] = [A]^{-1} = \begin{pmatrix} 1.5 & -2 \\ -0.5 & 1 \end{pmatrix}$$

2.13.18 MAT MLT Statement (Matrix Multiplication)

Function

The product of matrixes is calculated.

Format

```
MAT MLT(<Array Variable 1>(*),<Array Variable 2>(*),<Array Variable 3>(*))
```

Explanation

The product of the matrix specified by <Array Variable 2> and the matrix specified by <Array Variable 3> is calculated and assigned to the matrix specified by <Array Variable 1>.

The number of the rows of the matrix specified by <Array Variable 1> equal the number of the rows of <Array Variable 2>. The number of the columns of <Array Variable 1> must equal the number of the columns of <Array Variable 3>.

The multiplication of the elements of array variables is performed by the MAT MUL statement.

[Ex. 2.13.18-1]

The product of matrix B and matrix C is assigned to matrix A.

```
10   DIM A(2,2),B(2,3),C(3,2)
 :
50   MAT MLT(A(*),B(*),C(*))
 :
```

Assuming that: $[B]=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$, $[C]=\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$

then on line 50:

$$[A]=\begin{pmatrix}1\times1+2\times3+3\times5 & 1\times2+2\times4+3\times6\\ 4\times1+5\times3+6\times5 & 4\times2+5\times4+6\times6\end{pmatrix}=\begin{pmatrix}22 & 28\\ 49 & 64\end{pmatrix}$$

2.13.19 MAT MAX Statement (Matrix Maximum)

Function

This statement searches the maximum value of the elements of a matrix.

Format

MAT MAX(<Variable>, <Array Variable>(*))

Explanation

The maximum value of the elements of a matrix specified by <Array Variable> is assigned to <Variable>.

[Ex. 2.13.19-1]

The maximum value of the elements of the matrix A is assigned to X.

```
10   DIM A(3,4)
 :
50   MAT MAX(X,A(*))
 :
```

If $[A] = \begin{pmatrix} 6 & 8 & 7 & 8 \\ 2 & 1 & 9 & 3 \\ 10 & 2 & 1 & 4 \end{pmatrix}$

10 is assigned to X on line 50.

2.13.20 MAT MIN Statement (Matrix Minimum)

Function

This statement searches the minimum value of the elements of a matrix.

Format

```
MAT MIN(<Variable>, <Array Variable>(*))
```

Explanation

The minimum value of the elements of a matrix specified by <Array Variable> is assigned to <Variable>.

[Ex. 2.13.20-1]

The minimum value of the elements of matrix A is assigned to X.

```
10   DIM A(3,4)
 :
50   MAT MIN(X,A(*))
 :
```

If $[A] = \begin{pmatrix} 6 & 5 & 2 & 7 \\ -2 & 8 & 2 & -4 \\ 5 & 9 & 8 & 6 \end{pmatrix}$

-4 is assigned to X on line 50.

Matrix-related Instructions

No.	Instruction	Function
1	MAT INPUT (n, A(*))	Input to A from logical device number n
2	MAT READ (A (*))	Input to A from DATA statement
3	MAT PRINT (n, A (*))	Output of A to logical device number n
4	MAT MOV (A (*), B (*))	Assignment of B to A
	MAT MOV (A (*), Arithmetic Expression)	Assignment of arithmetic expression to A
5	MAT ADD (A (*))	Addition of B to A
	MAT ADD (A (*), Arithmetic Expression)	Addition of arithmetic expression to A
6	MAT SUB (A (*), B (*))	Substraction of B from A
	MAT SUB (A (*), Arithmetic Expression)	Subtraction of arithmetic expression from A
7	MAT MUL (A (*), B (*))	Multiplication A × B
	MAT MUL (A (*), Arithmetic Expression)	Multiplication A × arithmetic expression
8	MAT DIV (A (*), B (*))	Division A ÷ B
	MAT DIV (A (*), Arithmetic Expression)	Division A ÷ arithmetic expression
9	MAT SUM (X, A (*))	Sum of A
10	MAT CSUM (A (*), B (*))	Assignment of column sum to B from A
11	MAT RSUM (A (*), B (*))	Assignment of row sum to B from A
12	MAT IDN (A (*))	Construction of unit matrix of A
13	MAT INV (A (*))	Construction of inverse matrix of A
14	MAT TRN (A (*), B (*))	Assignment of transpose matrix of B to A
15	MAT DET (X, A (*))	Assignment of value of determinant of A to X
16	MAT MLT (A (*), B (*), C(*))	Product of matrixes
17	MAT MAX (X, A (*))	Assignment of maximum value of A to X
18	MAT MIN (X, A (*))	Assignment of minimum value of A to X

Note: 1. A, B, and C indicate array variables.
2. n indicates a logical device number.
3. X indicates an arithmetic variable.

3. Built-in Functions

3.1 Arithmetic Functions

Arithmetic functions are explained using arithmetic expressions X and Y.

3.1.1 EXP Function (Exponent)

Function

This function assigns the value of e^x (e = 2.7182818284586).

Format

```
EXP(<Arithmetic Expression>)
```

[Ex. 3.1.1-1]

The value of e^x is assigned to variable A.

```
 :
150 LET A=EXP(X)
 :
```

3.1.2 LOG Function (Log)

Function

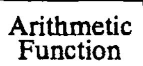
This function assigns natural logarithms.

Format

```
LOG(<Arithmetic Expression>)
```

Note

The value of <Arighmetic Expression> must be positive.



[Ex. 3.1.2-1]

The natural logarithm of X is assigned to variable A.

```
150 LET A=LOG(X)
```

3.1.3 LGT Function

Function

This function assigns common logarithms.

Format

```
LGT(<Arithmetic Expression>)
```

Note

The value of the < Artchmetic Expression > must be positive.

[Ex. 3.1.3-1]

The common logarithm of X is assigned to variable A.

```
 :
150 LET A=LGT(X)
 :
```

3.1.4 SQR Function (Square Root)

Function

This function assigns square roots.

Format

```
SQR(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be positive.

[Ex. 3.14-1]

The square root of X is assigned to variable A.

```
 :
150 LET A=SQR(X)
 :
```

3.1.5 FRC Function (Fraction)

Function

This function is extracts a fraction.

Format

```
FRC(<Arithmetic Expression>)
```

[Ex. 3.1.5-1]

The fractional part of the value of variable X is assigned to variable A.

```
 :
150 LET A=FRC(X)
 :
```

If X = 3.144592, A = 0.14592

3.1.6 RND Function (Random)

Function

This function generates random numbers.

Format

```
RND(<Arithmetic Expression>)
```

Explanation

If RND(X) is specified, a value determined by the value of X is generated. The value is generated as long as the value of X does not change. When the value of X is 0, random numbers determined by the value of the system clock are generated.

When RND is specified, new random numbers are generated based on the random numbers generated by the preceding RND function. When a RND function which has no parameter is used for the first time,the same result is obtained as when RND (0) is specified.

Note

Random numbers generated are 7-digit decimal fractions within the range: $0<X<1$.

[Ex. 3.1.6-1]

A random number determined by 8 is assigned to variable A.

```
 :
150 LET A=RND(8)
 :
```

[Ex. 3.1.6-2]

A random number based on the value of the system clock is assigned to variable A.

```
 :
150 LET A=RND(0)
 :
```

[Ex. 3.1.6-3]

A new random number is assigned to variable B based on the preceding random number.

```
 :
50  LET A=RND(0)
 :
100 LET B=RND
 :
```

3.1.7 ABS Function (Absolute)

Function

This function assigns an absolute value.

Format

```
ABS(Arithmetic Expression)
```

[Ex. 3.1.7-1]

The absolute value of X is assigned to variable A.

```
 :
150 LET A=ABS(X)
 :
```

3.1.8 SGN Function (Sign)

Function

This function checks the sign of a value.

Format

```
SGN(<Arithmetic Expression>)
```

Explanation

The value-1, 0 ro 1 is assigned depending on the sign of X.

X<0: SGN(X) = -1

X=0: SGN(X) = 0

X>0: SGN(X) = 1

[Ex. 3.1.8-1]

Program execution branches depending on the sign of X.

```
 :
150 ON SGN(X)+2 GOTO [MINUS],[ZERO],[PLUS]
 :
200 [MINUS] REM X<0
 :
250 [ZERO] REM X=0
 :
300 [PLUS] REM X>0
 :
```

3.1.9 FIX0 Function (Fix 0)

Function

This function rounds a fraction down to the specified decimal place.

Format

```
FIX0(<Arithmetic Expression>, <Variable>)
```

Explanation

The value of <Variable> is converted to an integer. Assuming that the integer is *n*, the value of <Arithmetic Expression> is rounded down to the *n*th decimal place.

[Ex. 3.1.9-1]

The value of X is rounded down to the third decimal place and the result is assigned to variable A.

```
:
50  LET A=FIXO(X,3)
:
```

If X=3.142592, A=3.141

3.1.10 FIX5 Function (Fix 5)

Function

This function rounds a decimal fraction off to the specified decimal place.

Format

```
FIX5(<Arithmetic Expression>, <Variable>)
```

Explanation

The value of <Variable> is converted to an integer. Assuming that the integer is *n*, the value of <Arithmetic Expression> is rounded of to the *n*th dicimal place.

[Ex. 3.1.10-1]

The value of X is rounded off to the third decimal place and the result is assigned to variable A.

```
:
50  LET A=FIX5(X,3)
:
```

IF X = 3.141592, A = 3.142

3.1.11 FIX9 Function (Fix 9)

Function

This function rounds a decimal fraction up to the specified decimal place.

Format

```
FIX9(<Arithmetic Expression>, <Variable>)
```

Explanation

The value of <Variable> is converted to an integer. Assuming that the integer is *n*, the value of <Arithmetic Expression> is rounded up to the *n*th decimal place.

[Ex. 3.1.11-1]

The value of X is rounded up to third decimal place and the result is assigned to variable A.

```
 :
50  LET A=FIX9(X,3)
 :
```

If X = 3.141592, A = 3.142

3.1.12 FIXE Function (Fix E)

Function

This function assumes a value of the E-type form and rounds its mantissa off to the specified number of digits.

Format

```
FIXE(<Arithmetic Expression>, <Variable>)
```

Explanation

When lagre numeric value, e.g. 123456789, is represented in E-type form, it becomes 1.23456789E8. The mantissa is rounded off to number of digits specified by the variable. The value of the variable is converted to an integer. Assuming that the integer is *n*, the value is rounded off to the *n*th decimal place. For example, if $n=3$, the value becomes 1.235E+8. That is, the original figure 123456789 is converted to 123500000.

[Ex. 3.1.12-1]

The mantissa of the value of X in the E form is rounded off to third decimal place and the result is assigned to variable A.

```
 :
50  LET A=FIXE(X,3)
 :
```

If X = 3.141592E+20, A = 3.142E+20.

3.1.13 INT Function (Integer)

Function

This function truncates a decimal fraction and assigns only an integer.

Format

```
INT(<Arithmetic Expression>)
```

[Ex. 3.1.13]

The integer part of the value of X is assigned to variable A.

```
 :
50  LET A=INT(X)
 :
```

If X = 123.456, A = 123

3.1.14 SIN Function (Sine)

Function

This function assigns a sine value.

Format

```
SIN(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be specified in degrees.

[Ex. 3.1.14-1]

The sine value of X is assigned to variable A.

```
 :
50 LET A=SIN(X)
 :
```

If X = 90, A = 1.

3.1.15 COS Function (Cosine)

Function

This function assigns a cosine value.

Format

```
COS(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be specified in degrees.

[Ex. 3.1.15-1]

The cosine value of X is assigned to variable A.

```
:
50  LET A=COS(X)
:
```

If X = 0, A = 1.

3.1.16 TAN Function (Tangent)

Function

This function is assigns a tangent value.

Format

```
TAN(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be specified in degrees. If a value is specified that makes the tangent value infinity (i.e. an integral multiple of ±90), the value of the function is the maximum or minimum value in the system.

[Ex. 2.1.16-1]

The tangent value of X is assigned to variable A.

```
:
50  LET A=TAN(X)
:
```

If X = 45, A = 1.

3.1.17 ASN Function (Arcsine)

Function

This function is assigns an arcsine value.

Format

```
ASN(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be specified as a numeric value. The result is within the range: $-90 \sim 90$ degrees.

3.1.18 ACS Function (Arccosine)

Function

This function is assigns an arccosine value.

Format

```
ACS(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be specified as a numeric value. The result is within the range: $0 \sim 180$ degrees.

[Ex. 3.1.18-1]

The arccosine value of X is assigned to variable A.

```
 :
50 LET A=ACS(X)
 :
```

If X = 1, A = 90

3.1.19 ATN Function (Arctangent)

Function

This function assigns an arctangent value.

Format

```
ATN(<Arithmetic Expression>)
```

Note

The value of <Arithmetic Expression> must be specified as a numeric value. The result is within the range: $-90 \sim 90$ degrees.

[Ex. 3.1.19-1]

The arc tangent value of X is assigned to variable A.

```
 :
50  LET A=ATN(X)
 :
```

If X = 1, A = 45.

3.1.20 RAD Function (Arctangent)

Function

This function converts degrees to radians.

Format

```
RAD(<Arithmetic Expression>)
```

Note

<Arithmetic Expression> must be specified in degrees. The result is within the range: $0 \sim 6.2831853071794$ (2π).

[Ex. 3.1.20-1]

The value of X is converted into radians and the result is assigned to variable A.

```
 :
50  LET A=RAD(X)
 :
```

If X = 180, A = 3.1415926535897.

3.1.21 DMS Function (Degree-Minute-Second)

Function

This function converts a decimal value in degrees to a sexagesimal value in degrees, minutes, and seconds.

Format

```
DMS(<Arithmetic Expression>)
```

Note

<Arithmetic Expression> must be specified in degrees. The decimal fraction consists of 4 digits. The first 2 digits indicate minutes and the last 2 digits indicate seconds.

[Ex. 3.1.21-1]

The value of X is converted into degrees, minutes, and seconds, and the result is assigned to variable A.

```
 :
50  LET A=DMS(X)
 :
```

If X=45.26, A=45.1536. This means that 45.26 degrees is equal to 45 degrees, 15 minutes, and 36 seconds.

3.1.22 ARD Function

Function

This function converts a radian value to a degree value.

Format

```
ARD(<Arithmetic Expression>)
```

Note

<Arithmetic Expression> must be specified in radians.

[Ex. 3.1.22-1]

Cos X is calculated in radians.

```
 :
50 LET A=COS(ARD(X))
 :
```

When X = 1.047 .... ($\frac{\pi}{3}$) radians, the value of ARD (X) is 60 degrees and A = 0.5.

3.1.23 ADS Function

Function

This function converts a value in degrees, minutes, and seconds into a value in degrees.

Format

```
ADS(<Arithmetic Expression>)
```

Note

Specify a decimal value for <Arithmetic Expression>. In the decimal number, the integer indicates degrees, the first 2 digits of the dacimal fraction indicate minutes, and the last two digits indicate seconds.

[Ex. 3.1.23-1]

The value of X is converted to a value in degrees and the result is assigned to variable A.

```
 :
50 LET A=ADS(X)
 :
```

If X = 45 degrees, 15 minutes, and 36 seconds (45.1536), A = 45.26 degrees.

3.1.24 MOD Function (Modulo)

Function

This function assigns the remainder of a division.

Format

```
MOD(<Arithmetic Expression 1>, <Arithmetic Expression 2>)
```

Explanation

The value of <Arithmetic Expression 1> is divided by the value of <Arithmetic Expression 2>, and the function value becomes the remainder of this calculation. When the value of <Arithmetic Expression 1> is negative, the remainder is also negative. When the value of <Arithmetic Expression 2> is greater than that of <Arithmetic Expression 1>, the remainder is the value of <Arithmetic Expression 1>.

[Ex. 3.1.24-1]

The remainder when X is divided by Y is assigned to the variable A.

```
 :
50 LET A=MOD(X,Y)
 :
```

The values of X, Y, and A are shown below.

X	Y	MOD (X, Y)
7	2	1
−10	3	−1
−10	− 3	−1
2	10	2

3.1.25 MAX Function (Maximum)

Function

This function assigns the maximum value from a specified group of variables.

Format

```
MAX(<Variable>, ···)
```

[Ex. 3.1.25-1]

The maximum value from the values of variables D (1) ~ D (4) is assigned to variable A.

```
:
50 LET A=MAX(D(1),D(2),D(3),D(4))
:
```

Assuming that D (1) = 130, D (2) = 2790, D (3) = 4, and D (4) = 2789, then A = 2790.

3.1.26 MIN Function (Minimum)

Function

This function assigns the minimum value from a specified group of variables.

Format

```
MIN(<Variable>, ···)
```

[Ex. 3.1.26-1]

The minimum value of the values of variables D (1) ~ D (4) is assigned to variable A.

```
 :
50 LET A=MIN(D(1),D(2),D(3),D(4))
 :
```

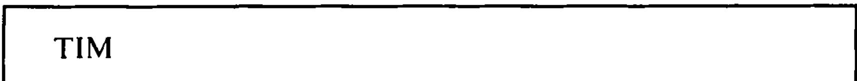
Assuming that D (1) = 130, D (2) = 2790, D (3) = 4, and D (4) = 5789, then A = 4.

3.1.27 TIM Function (Time)

Function

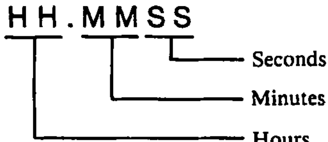
This function reads hours, minutes, and seconds from the AS-100's system clock.

Format



Explanation

The TIM function reads the time that has elapsed since power-on in hours, minutes and seconds. Hours, minutes and seconds are indicated as follows:



The value of the TIM function is within the range: 00.0000 ~ 23.5959. 00.0000 comes after 23.5959.

The value of the TIM function can be changed by the LET statement. The system clock used by the TIM function is the same as that used by string function TOD$.

[Ex. 3.1.27-1]

The processing time of an calculation routine is displayed.

```
 :
 50  LET TIME=0
 60  REM CALCULATION ROUTINE
 :
400  LET T=TIM
410  PRINT T
 :
```

TIM is set to 0 on line 50 and the time when the calculation routine ends is assigned to T on line 400.

For example, if 3.4513 is displayed, it means that calculation took 3 hours, 45 minutes, and 13 seconds.

3.1.28 PI Function (Pi)

Function

The function has a value of π (=3.1415926535898)

Format

```
PI
```

[Ex. 3.1.28-1]

The area of a circle whose radius is R is calculated and assigned to variable A.

```
:
50  LET A=PI*R**2  .
:
```

3.1.29 SIZE Function

Function

This function indicates the size of the unused memory area.

Format

SIZE

Explanation

The function has a value which indicates the size of the unused memory area when the function is executed. The value is in bytes.

[Ex. 3.1.29-1]

The size os the unused memory area when the program ends is displayed.

```
 :
910 PRINT SIZE
920 END
```

3.1.30 ERR Function (Error)

Function

This function detects a numeric value overflow.

Format

```
ERR
```

Explanation

There is a limit to the size of a value that can be processed by a system. (The limit is $9.9999999999999 \times 10^{63}$ for Canon BASIC.) An overfflow occurs if the result of a calculation exceeds this limit or a value larger than the limit is input. An overflow also occurs when a value is divided by 0.

When an overflow occurs, the value of the ERR function changes from 0 to 1. The ERR function is initially set to 0 when program execution begins.

The LET statement can be used to assign 0 to the ERR function.

[Ex. 3.1.30-1]

An overflow of a value is checked.

```
 :
 70  INPUT B,C
 :
150 LET A=B*C
160 IF ERR=1 GOTO 280
 :
280 PRINT "OVERFLOW"
290 LET ERR=0
300 GOTO 70
 :
```

When the result of the multiplication on line 150 exceeds the range: $1.0\times10^{-64}\leqq|x|<1.0\times10^{64}$, program execution branches to line 280. After "OVERFLOW" is displayed, processing is performed again from line 70.

On line 290, the overflow status of the ERR function is reset by assigning 0 with the LET statement.

3.1.31 EOF Function (End of File)

Function

This function detects the end of a specified file.

Format

```
EOF(<Arithmetic Expression>)
```

Explanation

<Arithmetic Expression> specifies the logical device number of the file defined by the OPEN statement.

When the specified file is opened, 0 is antomatically assigned to the EOF function corresponding to the file.

For details of the status under which the value of EOF function becomes −1, refer to the explanation of the INPUT, INPUT USING, and GET statements.

[Ex. 3.1.31.—1]

Data is read and displayed by random access from a file prepared by the PUT statement.

```
 10 OPEN #1,"B:FILE1"
  :
 50 INPUT RNO
  :
100 GET #1,RNO RECORD
110 IF EOF(1) GOTO 50
120 PRINT RECORD
150 CLOSE #1
  :
```

The record number displayed on line 50 is entered through the keyboard. If the record number entered is outside the file area, reinput must be made.

3.1.32 %CURX Function (%Cursor X)

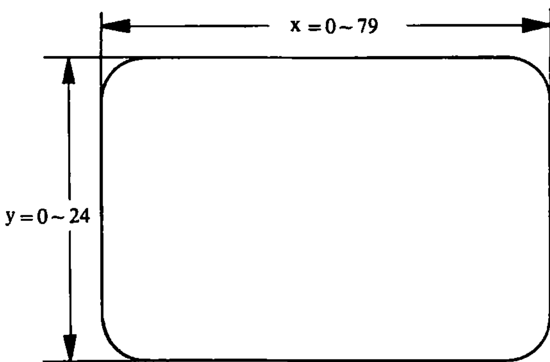
Function

This function reads the x-coordinate at the cursor position on the display.

Format

Explanation

This function has the value that indicates the x-coordinate of the cursor position on the display. The value is within the range: 0 ~ 79.



3.1.33 %CURY Function (%Cursor Y)

Function

This function reads the y-coordinate at the cursor position on the display.

Format

```
% CURY
```

Explanation

This function has the value that indicates the y-coordinate of the cursor position on the display. The value is within the range: 0 ~ 24.

Arithmetic Functions

No.	Functions	Use	Significant digits
1	EXP (X)	e^x	12
2	LOG (X)	loge X: natural logarithm	12
3	LGT (X)	$\log_{10}$ X: common logarithm	12
4	SQR (X)	$\sqrt{x}$: square root	14
5	FRC (X)	Extracting a decimal fraction	14
6	RND (X)	Random numbers	7
7	ABS (X)	$\|X\|$: absolute value	14
8	SGN (X)	Sign discrimination	1
9	FIX0 (X, Y)	Round-down	14
10	FIX5 (X, Y)	Round-off	14
11	FIX9 (X, Y)	Round-up	14
12	FIXE (X, Y)	E-type expression round-off	14
13	INT (X)	In integers	14
14	SIN (X)	Sin X	12
15	COS (X)	Cos X	12
16	TAN (X)	Tan X	12
17	ASN (X)	Sin^{-1} X	12
18	ACS (X)	Con^{-1} X	12
19	ATN (X)	Tan^{-1} X	12
20	RAD (X)	Degree → radians	14
21	DMS (X)	Degree → degrees, minutes, seconds	14
22	ARD (X)	Radians → degrees	14
23	ADS (X)	Degrees, minutes, seconds → degrees	14
24	MOD (X, Y)	Remainder of X ÷ Y	14
25	MAX (X_1, ···, X_n)	Maximum value	14
26	MIN (X_1, ···, X_n)	Minimum value	14
27	TIM	Hours, minutes, seconds	—
28	PI	π	12
29	SIZE	Size of unused memory area	—
30	ERR	Overflow check	1
31	EOF (X)	End of file checking	1
32	%CURX	x-coordinate of cursor	2
33	%CURY	y-coordinate of cursor	2

3.2 String Functions

3.2.1 LEN Function (Length)

Function

This function checks the number of characters in a character string.

Format

```
LEN (<String Expression>)
```

Explanation

This function has the value of the number of characters in a specified string expression. Each blank space is counted as one character.

[Ex. 3.2.1-1]

When X$ = "ABCDEFG", the number of characters in X$ is assigned to variable I.

```
  :
50  LET I=LEN(X$)
  :
```

When line 50 is executed, 7 is assigned to variable I. If X$ = "ABC␣␣EFG", I = 8. (␣ indicates one space.)

3.2.2 IDX Function (Index)

Function

This function confirms whether there is a specified character string in a certain string expression or not.

Format

```
IDX(<String Expression 1>, <String Expression 2>[, <Arithmetic Expression>])
```

Explanation

This function searches the character string specified by <String Expression 2> in a string expression specified by <String Expression 1>. If it is found, this function has a value indicating where the match lies from the beginning of <String Expression 1>. When there is more than one match in <String Expression 1>, the value of <Arithmetic Expression> specifies which occurrance of the match will be searched.

If the character string specified is not found, the value of this function is 0.

A space is counted as one character.

[Ex. 3.22-11]

When X$ = "RED␣WHITE␣BLUE␣WHITE␣YELLOW", the second character string "WHITE" is searched.

```
:
50 LET Z=2
60 LET A=IDX(X$,"WHITE",Z)
:
```

When line 150 is executed, character string "WHITE" is searched in string variable X$. Because the second "WHITE" starts at the 16th character, the value of A is 16.

3.2.3 VER Function (Verify)

Function

This function verifies that a certain character string consists only of the specified characters.

Format

```
VER(<String Expression 1>, <String Expression 2>)
```

Explanation

This function verifies that the character string specified by <String Expression 1> consists only of the characters specified by <String Expression 2>. Two or more characters can be specified for <String Expression 2>.

When <String Expression 1> consists only of the characters specified by <String Expression 2>, the value of this function is 0.

If <String Expression 1> contains a character not specified by <String Expression 2>, this function has a value indicating the column number where the unspecified character in <String Expression 1>.

[Ex. 3.2.3-11]

When X$ = "ABCCABC2B4A", find out at what column number in X$ a character other than A, B, or C appear for the first time.

```
:
50 LET A=VER(X$,"ABC")
:
```

When line 50 is executed, a check is performed to verify if there is any character other than A, B, or C in character string X$. Because the eighth character in X$ is 2, 8 is assigned to A.

3.2.4 NUM Function (Number)

Function

This function converts a numbers (character data) to a numeric value.

Format

```
NUM(<String Expression>)
```

Explanation

Some character strings consist of numbers (0~9) and look like numerical values, but they cannot be used for calculation. The NUM function converts such character data into numeric values. The following characters can be specified in a string expression:

- Numbers: 0~9
- Decimal point: . (If there is more than one decimal point, the characters following the second one are ignored.)
- Sign: + or − (The sign can be specified only at the beginning of numbers or immediately following E.)
- Exponent expression: E (If there is more than one E, the characters following the second E are ignored.)

If a character other than one of those listed above is included in the character string, all characters following that character are ignored.

If the string expression does not contain a number, the value of the function is 0.

[Ex. 3.2.4-1]

X$ is converted to a numeric value when X$ = "123AB5".

```
:
50 LET A=NUM(X$)
:
```

When line 50 is executed, the value 123 is assigned to A. "AB5" is ignored.

3.2.5 CHR$ Function (Character)

Function

This function converts a numeric value to character data.

Format

```
CHR$(<Arithmetic Expression>)
```

Explanation

The function is the opposite of the NUM function. It converts a numeric value specified by <Arithmetic Expression> to a string of numbers (character data).

This function has a defined length according to the number of digits. The first charcter is always a space.

If a number whose value is expressed in the E form is specified, it is converted into a character string in the E form.

[Ex. 3.2.5-11]

The character string "1234" is assigned to A$ when X = 1234.

```
:
50 LET A$=CHR$(X)
:
```

When line 50 is executed, the value 1234 is converted to a charater string and "␣1234" is assigned to A$.

3.2.6 ASC$ Function (ASCII)

Function

This function converts a specified ASCII code to character data.

Format

```
ASC$(<Arithmetic Expression>)
```

Explanation

This function's value is a one-byte character data in ASCII code that corresponds to the value specified by <Arithmetic Expression>. The numeric value in an arithmetic expression must be specified in decimal notation. The range of values that can be specified is : 0~255 (00~FF in hexadecimal notation). Specifying a value outside this range causes an error.

Refer to "Appendix 1. Character Codes" for ASCII codes.

[EX. 3.2.6-1]

The integer 65 is converted into a corresponding character.

```
 :
50 LET X=65
60 LET A$=ASC$(X)
 :
```

On line 60, character "A", which corresponds to the ASCII code 65 (41_H), is assigned to A$.

3.2.7 COD Function (Code)

Function

This function converts a specified character to ASCII code.

Format

COD(<String Expression>)

Explanation

This function has the value of the first character of a specified character string converted to decimal ASCII code.

Refer to "Appendix 1. Character Codes" for ASCII codes.

[Ex. 3.2.7-11]

When string variable X$ is "BOOK", the ASCII code that corresponds to the first character is assigned to variable A.

```
:
50 LET A=COD(X$)
:
```

When line 50 is executed, the first character (B) of X$ = "BOOK" is converted to ASCII code and 66 (42_H) is assigned to A.

3.2.8 STR$ Function (String)

Function

This function extracts part of a variable.

Format

```
STR$ (<String Expression>, <Arithmetic Expression 1>[, <Arithmetic Expression 2>])
```

Explanation

A character string specified by <Arithmetic Expression 1> and <Arithmetic Expression 2> is extracted from the character string specifed by <String Expression>.

<Arithmetic Expression 1> specifies the position of the character string that will be extracted.

<Arithmetic Expression 2> specifies the number of characters that will be extracted. <Arithmetic Expression 2> can be omitted. If it is omitted, all characters following the column specified by <Arithmetic Expression 1> are extracted.

The contents of <String Expression> do not change when a character string is extracted using this function.

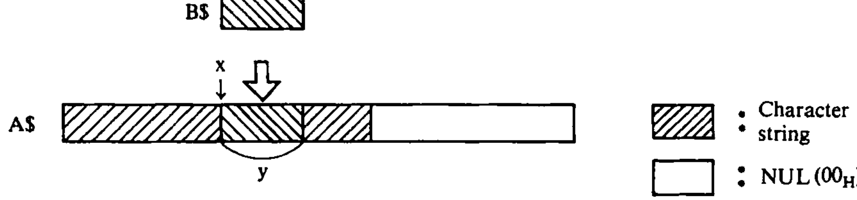
If this function is specified for the left side of an expression in the LET statement, a part of a specified character string can be replaced by another character string. In this case, the position of the first character of the character string that will be replaced is specified by <Arithmetic Expression 1>, and the number of characters that will be replaced is specified by <Arithmetic Expression 2>. <Arithmetic Expression 2> can be omitted. If it is omitted, all characters following the column specified by <Arithmetic Expression 1> are replaced.

For example, suppose that the following specification is made by the LET statement:

```
LET STR$(A$,X,Y)=B$
```

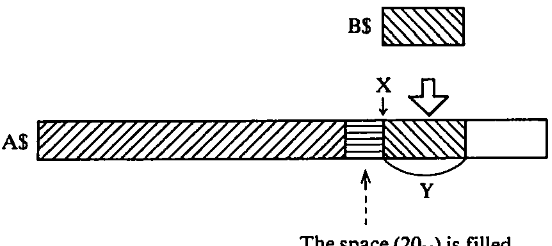
At this time, there are three different methods of replacement depending on the position specified by X:

- When the position specifed by X is in the character string of A$.



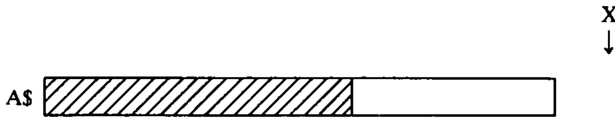
In this case, only the character string specified by Y is replaced. If fewer characters are specified by Y than are specified by B$, Y characters from the beginning of B$ are used to replace A$. If more characters are specified by Y than are specified by B$, all characters of B$ are used to replace A$, and the number of characters specified by Y is ignored.

- When the position specified by X exceeds the character area of X$:



Only the character string specified by y is replaced (added) and space codes are filled.

- When the position specified by X exceeds the character area of A$:



In this case, replacement is not performed.

[Ex. 3.2.8-1]

"DEF" is extracted from string variable X$="ABCDEFGHI" and assigned to string variable A$.

```
 :
10  DIM X$9
 :
50  LET A$=STR$(X$,4,3)
 :
```

When line 50 is executed, three characters starting from the fourth character are extracted from "ABCDEFGHI" and assigned to A$ so A$="DEF".

[EX. 3.2.8-2]

"DEF" in character string "ABCDEFGH" is replaced by "123".

```
 :
50  LET A$="ABCDEFGH"
60  LET STR$(A$,4,3)="123"
 :
```

When line 60 is executed, the contents of A$ changes from "ABCDEFG" to "ABC123GH".

In this case, even if "12345678" is specified instead of "123", the result is the same. If "12" is specified instead of "123," the result is "ABC12FGH".

[Ex. 3.2.8-3]

Assuming that "AB" is in a string variable with a length of 10 bytes, "123456" is added starting from the the fourth character.

```
10  DIM A$10
20  LET A$="AB"
 :
50  LET STR$(A$,4,6)="123456"
 :
```

When line 50 is executed, the contents of A$ changes from "AB" to "AB␣123456". B is followed by one space (20_H), and 6 is followed by a NUL code (00_H).

3.2.9 INPUT$ Function

Function

This function reads a character string.

Format

```
INPUT$ (<Arithmetic Expression 1>[, <Arithmetic Expression 2>[, <String Expression>]])
```

Explanation

This function reads character strings entered through the keyboard, character data from a file on a disk, or character strings output from an external input device.

This function is almost the same as the INPUT statement. But it has the values of the character strings read by the function itself. When the character (code) specified as the end code is read during the reading of a character string, the reading ends. <Arithemetic Expression 1> specifies the length (the number of characters) of the character string that will be read. Integers within the range: 1 ~ 255 can be specified.

<Arithmetic Expression 2> specifies the logical device number of the external input device or file from which reading will be performed. An integer of 0 ~ 9 can be specified. The logical device number must be defined in advance by the OPEN statement. But logical device number 0 indicates the keyboard, which is automatically opened by the system. <Arithmetric Expression 2> can be omitted. If it is omitted, the <String Expression> must also be omitted. It this case, the keyboard is automatically specified.

<String Expression> specifies the character that ends the reading. When this character is read during character string reading reading ends.

Up to four such characters can be specified as end codes at the same time. If more than one character is specified, reading ends when one of them is read. At the end of data reading, the end code itself is read. Even if an end code is specified, when the character string of the length specified by <Arithmetic Expression 1> is read, the reading ends. <String Expression> can be omitted. If it is omitted, the character string of the length specified by <Arithmetric Expression 1> is read.

"00_H" can not be specified as a end code.

Note

When a character string entered through the keyboard is read by this function, the characters stored in the key buffer are actually read and cleared from the key buffer.

[Ex. 3.2.9-1]

A string of 255 characters is read from the file on a disk. Reading ends when ":" is read.

```
:
10  DIM A$255
20  OPEN #1,"B:FILE1"
:
50  LET A$=INPUT$(255,1,":")
:
```

When the data that will be read from the file is "ABCDEFGH:IJI", A$ is "ABCDEFG:".

[Ex. 3.2.9-2]

A character string entered through the keyboard is assigned to string variable A$. Reading ends when a code representing a comma (,) or ↵ is read.

```
10  DIM A$128
:
50  LET A$=INPUT$(128,0,",&0D")
:
```

On line 50, the 0 is specifed for reading through the keyboard. Because the keyboard is already opened by the system, the OPEN statement does not have to be excuted.

When line 50 is executed, nothing on the screen but the system is set to the key input waiting state. Then if "ABCD," is entered, A$ becomes "ABCD,". Then execution proceeds to the next line and the key buffer is cleared. (Unlike the INPUT or INPUT USING statement, the INPUT$ function does not cause input echo back.)

3.2.10 KEY Function (Key)

Function

This function checks a character string entered through the keyboard.

Format

KEY[(<String Expression>)]

Explanation

Characters entered through the keyboard are first stored in the key buffer. The key buffer can store up to 128 characters.

With this function, the number of characters stored in the key buffer or and where character is located in the key buffer can be checked. Only one character can be specified in <String Expression>. If more than one character is specified, only the first character is valid, and other characters are ignored. This function has a value indicating the specified character's column number in the character string stared in the key buffer. If the specified character is not found, the value is 0.

If <String, Expression> is omitted, this function has the value indicating the number of characters in the buffer. A value of 0 ~ 128 is assigned to the function.

Note

The contents of the key buffer remain as long as they are not read by the INPUT$ function. So, clear the buffer before the program ends. Refer to the example to learn how to clear the buffer.

[Ex. 3.2.10-1]

Characters entered through the keyboard are stored in the key buffer. Processing starts when ↵ is depressed.

```
 5  DIM A$10
10  IF KEY >=10 GOTO [EXIT]
20  IF KEY("&0D")<>0 GOTO [EXIT]
30  GOTO 10
40  [EXIT] REM PROCESSING START
50  PRINT "PROCESSING START"
 :
```

A loop is made by lines 10~30. The branch condition from this loop is the storage of 10 characters in the key buffer or the depression of ↵ . "&0D" on line 20 is the code for ↵ .

When this program is executed, nothing is displayed and the system is set to the key input waiting state. Character can be entered through the keyboard in this state. The character string is not displayed on the screen but is stored in the key buffer. When ↵ is depressed or 10 characters are entered, "PROCESSING START" is displayed on the screen.

[Ex. 3.2.10-2]

The contents of the key buffer are cleared.

```
 :
50  A$=INPUT$(KEY)
 :
```

When line 50 is executed, the value of the KEY function (the number of characters in the buffer) is used as the parameter for the INPUT$ function and the contents of key buffer are assigned to A$. So the key buffer is cleared.

[Ex. 3.2.10-3]

When a comma (,) is entered, the character string is assigned to A$.

```
 5  DIM A$128
10  IF KEY(",")=0 GOTO 10
20  A$=INPUT$(KEY(","))
30  A$=STR$(A$,1,LEN(A$)-1)
 :
```

On line 10, the input through the keyboard is stored in the key buffer. When the comma is entered, the value of KEY changes from 0 to the number of characters stored in the key buffer and execution proceeds to line 20.

On line 20, the entire character string (including the comma) in the key buffer is assigned to A$.

On line 30, the entire character string except the comma is extracted and assigned again to A$.

When the program is executed, nothing is displayed and the system is set to the input waiting state. At this time, if "ABCDE," is entered through the keyboard, "ABCD" is assigned to A$.

3.2.11 FKEY Function (Function Key)

Function

This function checks a key when it is depressed.

Format

```
FKEY[(<Arithmetic Expression>)]
```

Explanation

Function keys and other special keys generate the character strings shown in Table 3.1 in the key buffer.

<Arithmetic Expression> specifies the number corresponding to a key shown in Table 3.1. The range of values is: 1 ~ 43.

When <Arithmetic Expression> is specified, the function searches the key buffer for the character string corresponding to the specified key. If the character string is found, the function has a value indicating the number of column where the character string begins. At the same time, the character string is cleared from the key buffer and the remaining character string is moved up. If the character string corresponding to the specified key is not found, the value of the function is 0 and the contents of the key buffer do not change.

If <Arithmetic Expression> is not specified, the function checks if there is a character string at the head of the key buffer that corresponds to any of the keys shown in Table 3.1. If such a character string is found, the function has a value indicating the the key number in the table. At this time, the character string found is cleared from the key buffer and the remaining character string is moved up. If there is not a character string corresponding to a key in the table, the value of the function is 0 and the contents of the key buffer do not change.

Table 3.1

No.	Key	Key No.	Character String	No.	Key	Key No.	Character String
1	F1	1	ESC O	23	→(6)	23	ESC [C
2	F2	2	ESC P	24	←(4)	24	ESC [D
3	F3	3	ESC Q	25	PgUp (9)	25	ESC [E
4	F4	4	ESC R	26	PgDn (3)	26	ESC [F
5	F5	5	ESC S	27	—(5)	27	ESC [G
6	F6	6	ESC T	28	HOME (7)	28	ESC [H
7	F7	7	ESC U	29	—(1)	29	ESC [I
8	F8	8	ESC V	30	CLEAR SCREEN	30	ESC [2J
9	F9	9	ESC W				
10	F10	10	ESC X	31	—(0)	31	ESC [N
11	F11	11	ESC Y	32	⇧/F1	32	ESC o
12	F12	12	ESC Z	33	⇧/F2	33	ESC p
13	COPY	13	ESC 3	34	⇧/F3	34	ESC q
14	MOVE	14	ESC 4	35	⇧/F4	35	ESC r
15	DELETE	15	ESC 5	36	⇧/F5	36	ESC s
16	INSERT	16	ESC 6	37	⇧/F6	37	ESC t
17	⇧ ⇆	17	ESC 7	38	⇧/F7	38	ESC u
18	PD-A	18	ESC 0	39	⇧/F8	39	ESC v
19	PD-B	19	ESC 1	40	⇧/F9	40	ESC w
20	PD-C	20	ESC 2	41	⇧/F10	41	ESC x
21	↑(8)	21	ESC [A	42	⇧/F11	42	ESC y
22	↓(2)	22	ESC [B	43	⇧/F12	43	ESC z

Note

1. ESC indicates "$1B_H$" in ASCII code.
2. Numbers 0 ~ 9 in parentheses indicate ten-key numbers.
3. Numbers 21 ~ 29 and 31 indicate key operations performed in the cursor control mode.
4. – indicates that nothing is printed on a key.
5. ⇧ indicates that the key is depressed simultaneously with the shift key.
6. PD-A ~ PD-C are function keys on the pointing device.

[Ex. 3.2.11-1]

When [→] is depressed, execution proceeds to the next routine.

```
 :
50  IF FKEY(23)=0 GOTO 50
60  REM MOVE CURSOR TO RIGHT
 :
```

When [→] (key no. 23) is depressed on line 50, the next line is executed. At this time, character string "ESC [C" in the key buffer is cleared.

[Ex. 3.2.11-2]

When [F1], [F2], or [F3] is depressed, a branch occurs to the respective routine.

```
10  DIM A$1
 :
40  IF KEY=0 GOTO 40
50  ON FKEY GOTO [F1],[F2],[F3]
60  A$=INPUT$(1)
70  GOTO 40
 :
100 [F1] REM F1 PROCESSING
 :
150 [F2] REM F2 PROCESSING
 :
200 [F3] REM F3 PROCESSING
 :
```

The system awaits key input on line 40. If a key is depressed, it is checked on line 50. If the key is one of the function keys specified, a branch occurs to the respective processing. If the key is not one of the function keys specified, line 60 is executed to clear the key buffer, and the execution returns to line 40.

3.2.12 COM$ Function (Command)

Function

This function extracts a character string specified by the RUN command.

Format

```
COM$
```

Explanation

This function enables the use of a character string, entered after a semicolon (;) in the RUN command, in a program.

If there is no semicolon or character string following the semicolon, this function has NUL code (00_H) as its value.

Note

The semicolon immediately following the RUN command is not included in the character string of this function.

[Ex. 3.2.12-1]

In the RUN command input to execute program EX1, the name of the file that will be used in the program is specified.

```
 5 REM   PROGRAM  EX1
10 DIM DF$10
 :
50 LET FNAME$=COM$
60 DF$="A:"+FNAME$
70 OPEN #1,DF$
 :
```

Assume that the following is input to start program execution.

RUN EX1;FABC ↵
EX1;FABC ↵
or EX1;FABC ↵
or ;FABC ↵

In each case, data file FABC. DAT on the disk in drive A is opened on line 60.

3.2.13 HEX$ Function (Hexadecimal)

Function

This function converts a specified value to a hexadecimal figure.

Format

```
HEX$(<Arithmetic Expression>)
```

Explanation

This function converts a value specified by <Arithemetic Expression> to a two-digit hexadecimal figure (character string). The range of values that can be specified by <Arithmetic Expression> is : 0~255.

[Ex. 3.2.13-1]

The ASCII code of a character entered through the keyboard is converted to a hexadecimal figure.

```
 5 DIM A$1,X$2
10 INPUT A$
20 LET Q=COD(A$)
30 LET X$=HEX$(Q)
40 PRINT X$
 :
```

If "$" is entered on line 10, ASCII code 36 (decimal) is assigned to Q on line 20. On line 30, character string 24, which is the hexadecimal figure equivalent to 36, is assigned to X$. So "24" is displayed on line 40.

3.2.14 TOD$ Function (Time of Day)

Function

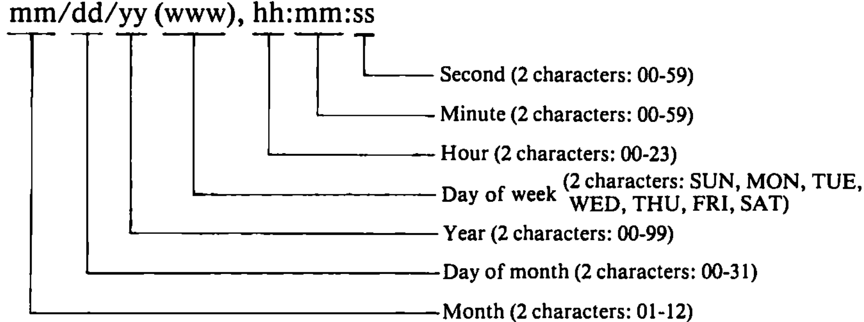
This function sets the date, day, and time.

Format

```
TOD$
```

Explanation

This function has the value of the AS-100's system clock. It is expressed as a 22-character string as shown below.



The value of the AS-100's system clock is set as the day and the time.

The value of the system clock can be also set by the OS mode command TOD.

The hour, minute, and second value handled by this function is linked to the TIM function.

- Method for setting the data and time
  TOD$ = "mm/dd/yy, hh:mm:ss"
- Method for setting the date only
  TOD$ = "mm/dd/yy"
- Method for setting the time only
  TOD$ = "hh:mm:ss"

When the date or the time is set, a two-digit figure must be specified for each element. When the date (year/month/day) is specified, the day of week is automatically set.

The value is counted according to the system clock.

Note

When the real time clock (option) is attatched to the AS-100, the value of the system clock elapses even when the power is off.

[Ex. 3.2.14-1]

January 5, 1983, 8:30:00 is set.

```
10 LET TOD$="01/05/83,08:30:00"
 :
```

[Ex. 3.2.14-2]

A date entered through the deyborad is set.

```
10 INPUT MSG("DATE(mm/dd/yy)=")DAY$
20 LET TOD$=DAY$
 :
```

[Ex. 3.2.14-3]

The value of the system clock is displayed.

```
 :
50 PRINT TOD$
 :
```

The result is as shown below. The contents is the value of the system clock at the time line 50 is executed.

"01/05/83 (WED), 08:35:12"

Note

When the system clock is set using the TOD command (an OS mode command), a space (␣) instead of a comma, must be placed between the data and the time as follows:

TOD mm/dd/yy␣hh:mm:ss

List of Character Processing Functions

No.	Function	Use
1	LEN (X$)	Number of characters
2	IDX (X$, Y$, Z)	Search
3	VER (X$, Y$)	Locate
4	NUM (X$)	Character string → numeric value
5	CHR$ (X)	Numeric value → character string
6	ASC$ (X)	ASCII code → character string
7	COD (X$)	Character → ASCII code
8	STR$ (X$, Y, Z)	Extraction and replacement
9	INPUT$ (X, Y, Z$)	Reading
10	KEY (X$)	Key buffer retrieval
11	FKEY (X)	Function key retrieval
12	COM$	Extracts the character string form RUN commands
13	HEX$ (X)	Decimal value → hexadecimal figure
14	TOD$	Date and time

4. ISAM Function

This section explains the ISAM function of Canon BASIC, which can be used by loading an ISAM library into memory.

4.1 What Is ISAM?

ISAM is an abbreviation for Indexed Sequential Access Method. This function reads and writes data (indexed access) to and from a file by referencing the contents of the data as an index.

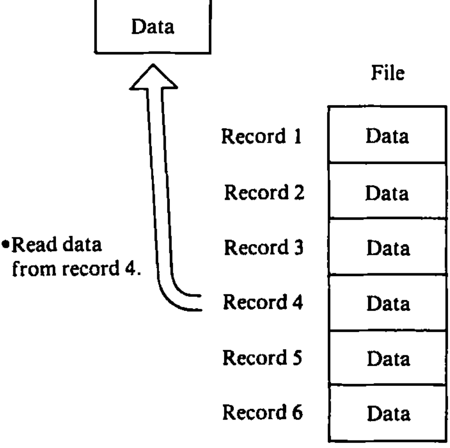
4.1.1 Indexed Access

When accessing a file using the GET statement, the PUT statement, etc., data is read from and written to the file by specifying the record number in the file.

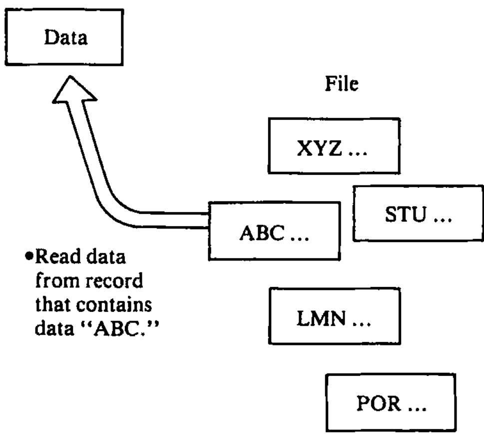
With indexed access, data is read from and written to a file by specifing the contents of records as an index, disregarding the record numbers in the file (the record positions in the file).

For example, with ordinary access, an instruction is given to read the data from record 4. But with indexed access, an instruction is given to read the data from the record that contains "ABC".

- Access with GET, PUT and others



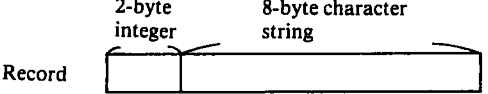
- Indexed Access



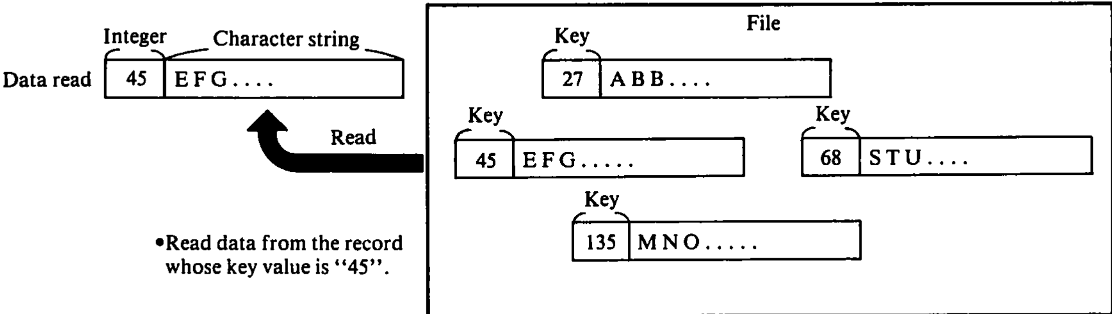
An index for indexed access is called a key. The part of the record where "ABC" is written is the key in the figure.

The key can be set at any position in a record. The data in the key (key value) of the record serves as the index in indexed access.

Assume that a record the following contents:



If the first two bytes are set as a key, indexed access can be performed using the value of the 2-byte integer data as an index. The record that has data "45" as a key can be read from the file by indexed access when the data "45" is specified as a key.

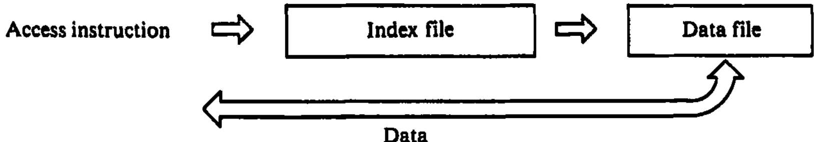


Indexed access is possible because there is an index file. An index file records the key information of the records in a data file. An index file is also created when a data file which is accessed with indexed access is created.

When data is read from a file with indexed access, the key information in the index file is first retrieved according to the specified key value. Then the record specified by the key information is read.

With this procedure the data of a specific record can be read into a program.

In indexed access, a data file and an indexed file function only when they are used together, so, they must always be copied together.

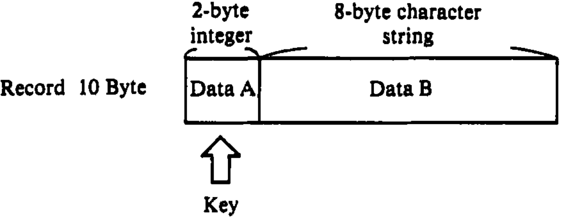


In indexed access, allocation of a data file and an index file to a disk, writing of key information into the index file, and other processing are all performed automatically by the ISAM function. So during design, only the data structure of the record and the location of the key have to be considered.

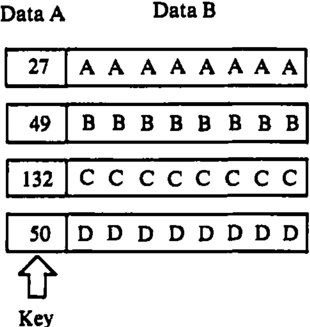
4.1.2 Keys

This part explains the rules governing a key as an access index in indexed access.

Assume that a record is as shown here.



And assume that a file consisting of four records with the following data is created with indexed access:



Because data A is defined as a key, the priority shown below is assigned to each record according to the key value. The priority is not related to the order of writing the records. The ISAM function automatically assigns higher priority to smaller key values by referring to the key values of the records in the file. This priority is called a key order.

	Data A	Data B
Key order 1	27	A A A A A A A A
Key order 2	49	B B B B B B B B
Key order 3	50	D D D D D D D D
Key order 4	132	C C C C C C C C
	⇧ Key	

Reading data from the records continuously in key order is called indexed sequential read.

Reading data from the records, without regard to the key order, by specifying the key values (Data A) of the records that will be read is called indexed random read.

4.2 Canon BASIC ISAM Function

This part describes how indexed access is performed by the ISAM function of Canon BASIC.

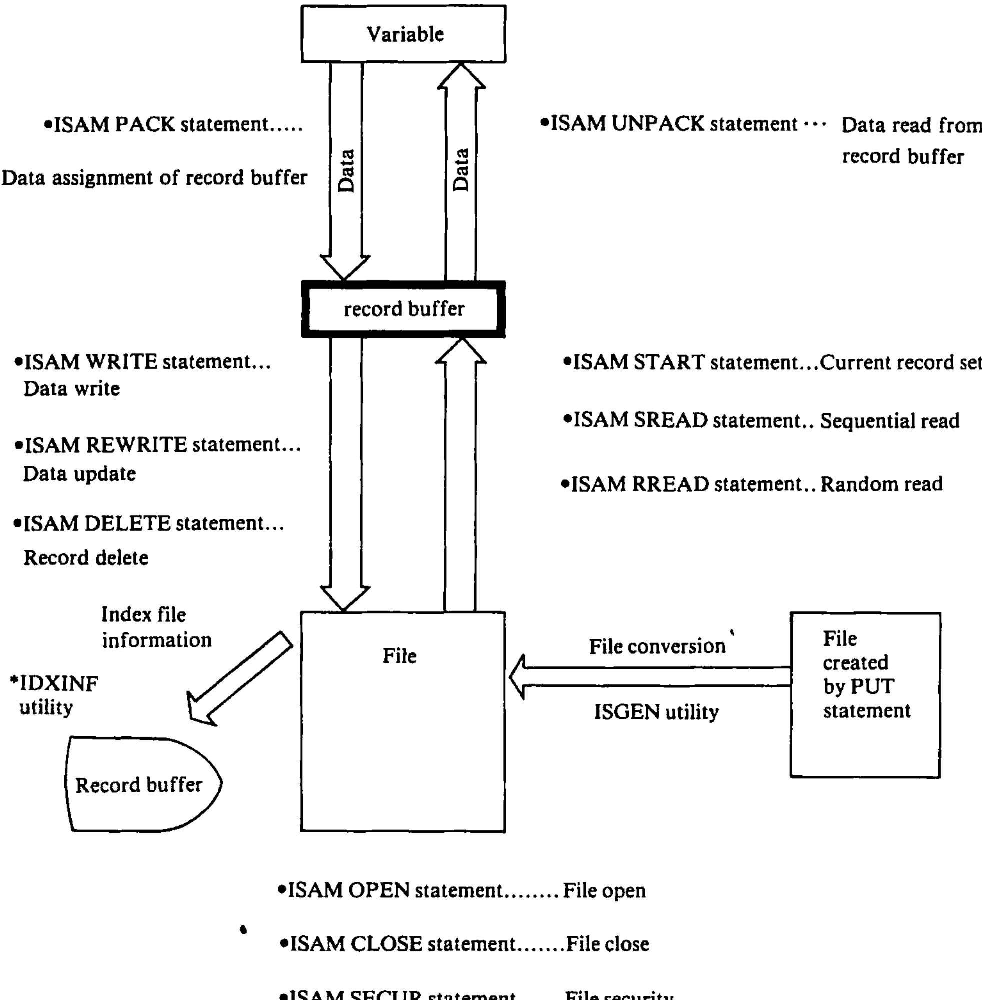
4.2.1 General

The ISAM function of Canon BASIC uses a special data file and a special index file for indexed access. These files have a different structure than data files used by the GET, PUT, and other statements. A file created by the PUT statement can be converted to a file for ISAM using ISGEN, one of the ISAM utility programs explained later.

A file created by ISAM can be accessed by the ISAM function of Level II COBOL*(Canon AS-100 Specifications).

The ISAM function is executed by eleven ISAM instructions and two ISAM utility programs. The functions of the instructions and the utilities are illustrated on the next page.

*Level II COBOL is a trademark of Micro Focus Ltd.



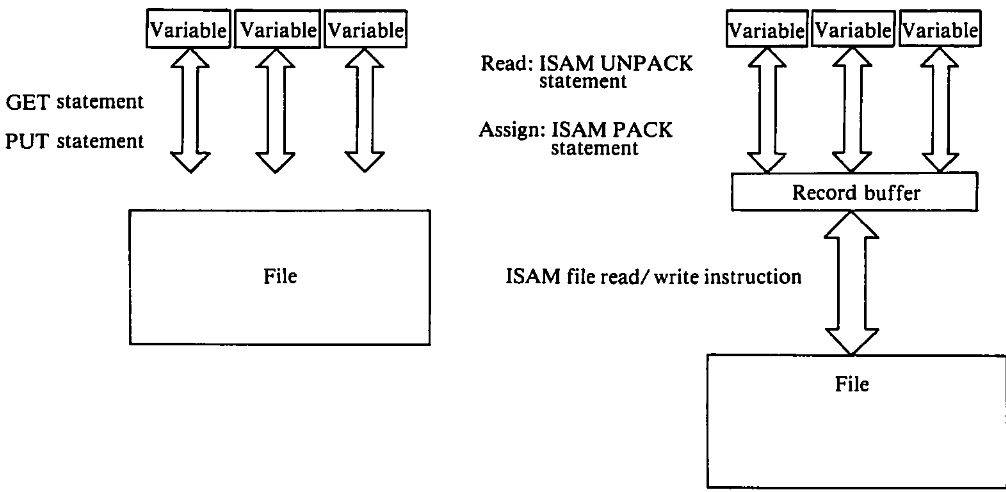
The functions of the instructions and utilities are explained in detail later.

4.2.2 Records

Although the record length is determined by the length of the variable specified in the GET statement and the PUT statement, it is regarded as fixed in ISAM processing. The record length can be defined within the range: 1 ~ 510 bytes. A fixed record length here means that all the record lengths in the same file are equal and the reading/writing of the data from/to file is always performed in record units.

In ISAM, data cannot be read and written directly. Data is first stored in an area called a record buffer before it is written into a file or assigned to a variable. For example, when one data record is written to a file, the value of each variable that consists of the record data is assigned to the record buffer in advance using the PACK statement. Then an instruction to write the contents of the record buffer into the file is executed.

•Ordinary file reading/writing •File reading/writing by ISAM



The record buffer consists of a record length of string-type array variables in which each array element is one byte. By using the PACK statement (assign) and the UNPACK statement (read), the record buffer delivers or receives data to/from the variables.

4.2.3 Primary Keys and Alternate Keys

Up to four keys in each record can be set in ISAM. There is one primary key and three alternate keys. The primary key must always be set.

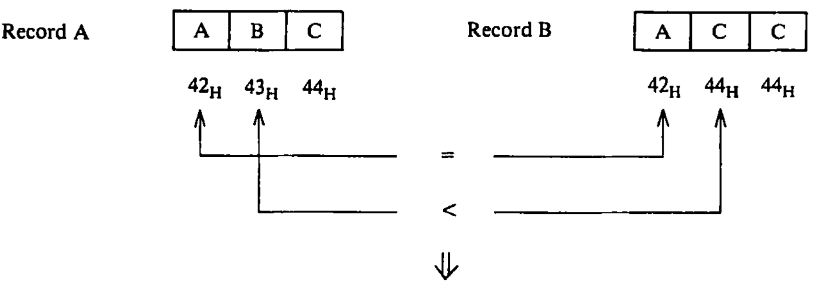
The value of the primary key cannot be duplicated in a file. This means that records having the same primary key value are not written to a file.

One, two, or three alternate keys can be set. It must be specified if duplication is valid or not for each key.

The length of each key is specified within the range: 1 ~ 32 bytes.

Key values are compared byte by byte from the beginning of the key regardless of the data type.

For example, when comparing character strings "ABC" and "ACC", both strings have "A" in the first byte. Comparison proceeds to the second byte, where "B" and "C" are compared. Because "B" = 42_H and "C" = 43_H, the key value "ACC" is regarded as greater than that of "ABC".



Record A < Record B

Key values are compared in the same way for integer type-data (2 bytes) and real number-type data (8 bytes).

Since the numeric order and the order of key value are different for negative values, negative numeric values should not be used in ISAM.

The comparative order of key values of integer-type data are as shown below.

Minimum 0 → 32767, −32768 → −1 Maximum

Note Refer to "Appendix 6, Execution of Machine Language Programs". for the structure of integer-type data and real number-type data.

4.2.4 Files

Regardless of files opened with the OPEN statement, under ISAM up to six pairs of files can be open at the same time.

Each file is opened with the ISAM OPEN statement. File numbers are defined automatically to the files. The file numbers must be specified to access these files in the program.

When a files are opened, the open mode (described later) must be specified to the opened files according to the mode access.

The following number of records can be written into a pair of files:

- When the record length is 126 bytes or less: 65535 records

- When the record length is 127 bytes or more:
  The value of the integer N in
  $(\text{Record length} + 2) \times N \leqq 65535 \times 128$

* These limits do not apply when the disk is full.

4.2.5 Pointer

When an index for indexed access is a key value, a pointer in ISAM serves as a direct index for file access.

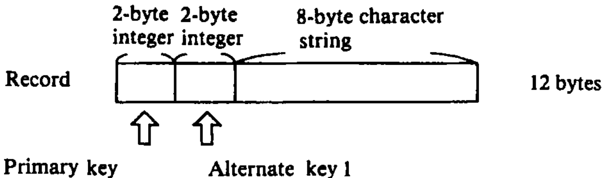
For example, assume that a key value is specified and the data of the record that has the key value specified is read.

ISAM first searches the key information in the index file for the specified key value and finds the corresponding record in the data file.

Then ISAM moves the pointer to the corresponding record, reads the data of the record indicated by the pointer, and assigns the data to the record buffer. After this processing, ISAM moves the pointer to the next record in the key order and processing is completed.

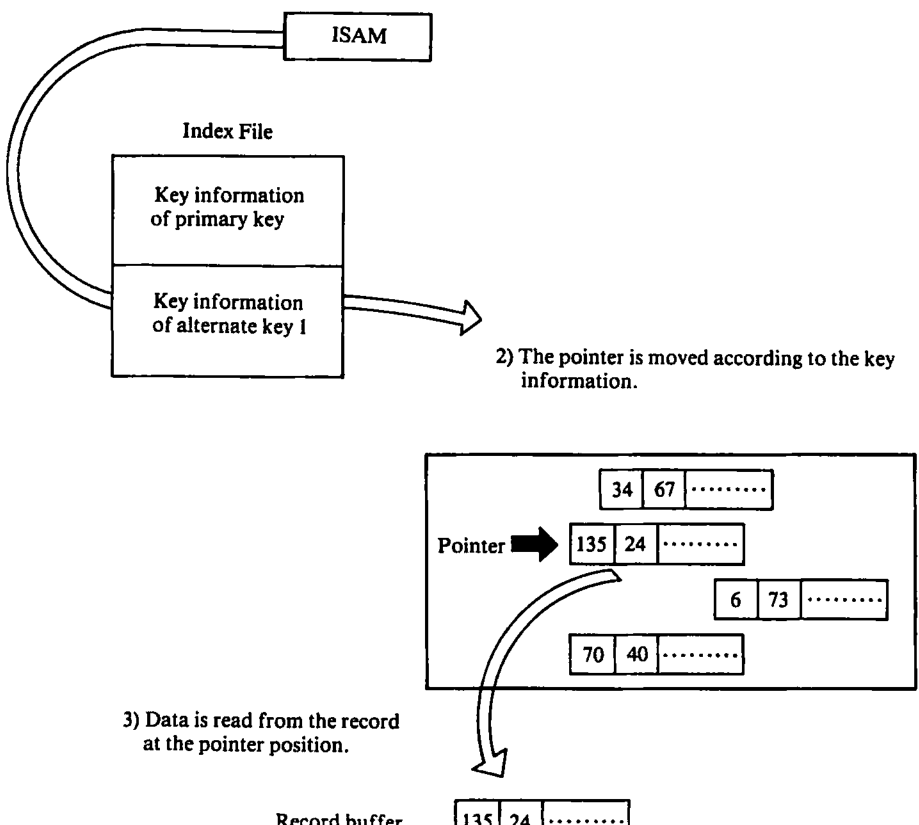
The record where the pointer is located is called the current record and the key serving as the basis of pointer movement is called the current key.

This process is shown here.

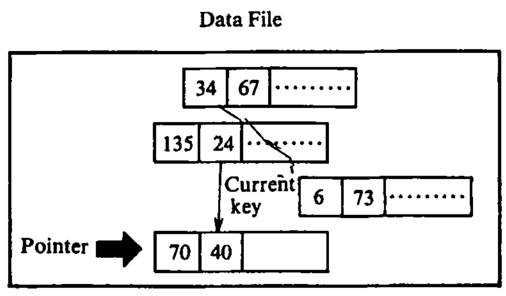


- Reading of data from the record in which the alternate key value is 24.

1) Key information of alternate key 1 is searched.



4) For alternate key 1, the pointer is moved to the record next in the key order.



5) End of processing

The current key has nothing to do with indexed random access, but it serves as an index for access in indexed sequential access. The next record in indexed sequential access means the next record in the key order with respect to the current key.

4.2.6 Limitations and Notes for Use

- Up to six files can be open at the same time.
- The record length is within the range: 1 ~ 510 bytes.
- The maximum number of keys is four: 1 primary key and 3 alternate keys.
- The maximum key length is 32 bytes.
- The maximum number of records that can be written in a pair of files is 65535.
  1) Record length $\leqq$ 126 . . . . . 65535 records
  2) Record length $\geqq$ 127 . . . . . N records in
     (Record length 2) $\times$ N $\leqq$ 65535 $\times$ 128
- The value of the primary key cannot be duplicated.
- Key values are compared byte by byte as characters. The order of a numeric value and the order of key values are different for negative values.
- The ISAM library must always be loaded at BASIC start-up and be resident in memory.

4.3 How To Use ISAM Instructions

This part describes the procedures for using ISAM.

4.3.1 Loading the ISAM Library

The ISAM library must always be resident in memory to use ISAM. "/ISAM" must be specified in the BASIC command to start BASIC, and the ISAM library is loaded into memory when the BASIC command is executed.

- Operation

BASIC ␣ /ISAM ↵

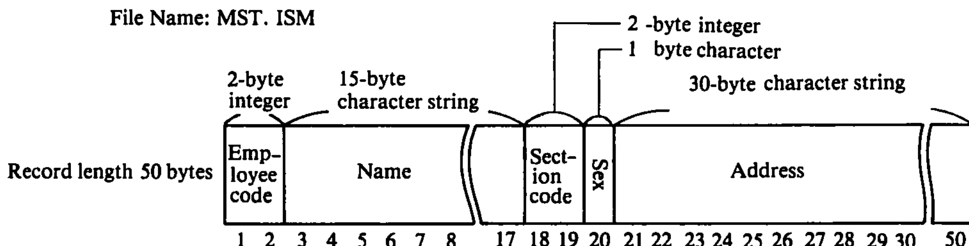
4.3.2 Design of Files

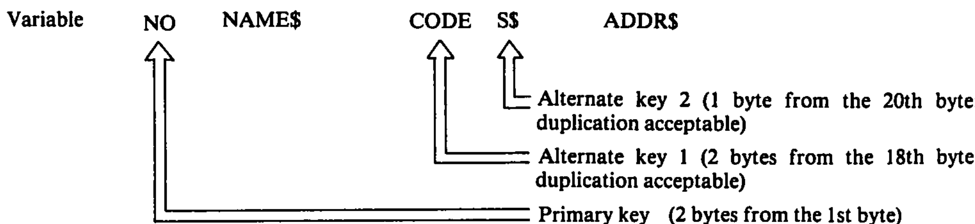
A file for indexed access must be designed for use by ISAM. Only the record structure must be considered because file design is determined by the record design.

- Data structure: The data that makes up a record is determined.
- Record length: The record length is determined according to the data structure of the record. It must be within the range: 1 ~ 510 bytes.
- Key: The key part of the record is determined.
- File name: The file type specified should be ISAM to prevent confusion with non-ISAM data files. The name of a data file added to the file type IDX is automatically specified as an index file.

*Refer to "4.10 How To Calculate File Size" to calculate the size of a file.

[Ex.]





4.3.3 Variables

The use of ISAM requires a number of different variables. The names of the variables must be defined in advance. The examples shown here are the same variables used in the ISAM program examples.

- Data .... Variables to which data making up records is assigned
  NO, NAME$, etc.

- Record buffer .... String-type array variables used as a record buffer Each element has a length of 1 byte.
  BUF$(1) ~ BUF$(n)

- File structure information .... Integer-type array variables to which data indicating the file structure specified for the ISAM OPEN statement are assigned
  PARM(1) ~ PARM(n)

- File number .... This is an integer-type variable to which a file number is assigned.
  ID

- Return code .... This is an integer-type variable to which a return code indicating the result of ISAM instruction execution is assigned.
  STAT

The details of these variables are explained in the explanation of each ISAM instruction. In addition to these variables, other variables necessary to execute of ISAM instructions must sometimes be defined.

4.3.4 Return Code

When an ISAM instruction is executed, a return code indicating the result of instruction execution is automatically assigned to a specified variable. Be sure to check this return code following execution of an ISAM instruction in the program.

4.3.5 How To Interpret Formats

Like other Canon BASIC statements, each ISAM statement consists of an instruction word (keyword) and an operand. Specify a keyword and an operand according to the format shown for each ISAM statement.

But the method of specifying formats is somewhat different from that of other instructions. In other Canon BASIC statements, a specifiable element such as <Arithmetic Expression> is shown in the operand specified in the format. In ISAM statements, the definition of an element in the operand is specified as <File Number>, <File Name>, etc.

The part that specifies the variable to which the data is returned from ISAM is underlined (==) in the format.

Example:

ISAM OPEN (<File No.>, <File Name>, <File Structure>, <Return Code>)

4.4 Basic ISAM Instructions

4.4.1 ISAM OPEN Statement (ISAM Open)

Function

This statement opens a pair of data files and index files.

Format

```
ISAM OPEN (<File No.>, <File Name>, <File Structure> ,
                                  <Return Code>)
```

Explanation

The ISAM OPEN statement opens a pair of data files and index files for indexed access.

Specify an integer-type variable in <File No.>. When this statement is executed, an identification number (1 ~ 6) of the pair of files opened is automatically assigned to the variable as the file number. Specify the file number assigned to the variable for indexed access to the file after execution of the statement. If the file is not opened, 0 is assigned.

Specify the name of the data file that will be opened with the drive name, file name, and file type in <File Name>.

It must be specified with a string expression (a string variable or a character string in quotation marks). An index file with the same name as the specified data file with the IDX file type attached, is opened by the ISAM OPEN statement on the same disk as the data file. If the drive name is omitted, the current drive is automatically specified.

Any file type can be specified here. If the file type is omitted, a file whose file type isn't defined is opened. Because the CANCEL command of the Canon BASIC cannot delete a file whose file type isn't defined, specify this file type as "ISM". A file created by ISAM cannot be accessed using the GET statement, PUT statement, etc. Do not use the file type "DAT" to prevent confusion with ordinary data files.

Specify an integer-type array variable of one dimention to which the value corresponding to the structure of the file that will be opened is assigned in advance in <File Structure>.

For the file structure (record information, key information, etc.) of the pair of files that will be opened using the ISAM OPEN statement for indexed access, the value corresponding to the structure must be assigned in advance to the integer-type array variable of one dimension by the method shown below.

For the integer-type array variable, the length corresponding to the quantity of the file structure information must be defined in advance with the INTEGER and DIM statements.

Specify the file structure information as shown below.

	Subscript of array		
	1	Open mode	Input mode: 1, Output mode: 2, Update Mode:3
	2	Record length	1 ~ 510
	3	Primary key position	1 ~ Record length
	4	Primary key length	1 ~ 32
Omissible	5	Attribute of alternate key 1	Duplication unacceptable: 1, duplication acceptable:2
Omissible	6	Position of alternate key 1	
Omissible	7	Length of alternate key 1	
Omissible	8	Attribute of alternate key 2	
Omissible	9	Position of alternate key 2	
Omissible	10	Length of alternate key 2	
Omissible	11	Attribute of alternate key 3	
Omissible	12	Position of alternate key 3	
Omissible	13	Length of alternate key 3	
	14	0	End code

- Open Mode

Select the input mode (1), output mode (2), or update mode (3) according to the form of indexed access.

Specify the input mode when only the reading of data from the file will be performed. When files are opened in this mode, data cannot be written to the file. An error occurs if the pair of files specified in this mode are not on the disk.

Specify the output mode to create a file. If files are opened in this mode, data cannot be read from the file. When files are opend in this mode, if the specified files are already on the disk, they are deleted from the disk and new files are created.

Specify the update mode to read data from a file and write data to a file. When files are opened in this mode, if the specified files are not on the disk, new files are created. An error occurs if either the data file or an index file specified is not on the disk.

- Record Length

  Specify the length of the record in bytes. The length must be an integer within the renge: 1 ~ 510.

- Primary Key Position

  Specify 1 plus the number of bytes preceding the position of the primary key. With the beginning of the record being 1, specify an integer within the range of the record length.

- Primary Key Length

  Specify the primary key length in bytes. Specify an integer within the range: 1 ~ 32. Make sure that the key range does not exceed the record length.

- Attribute of Alternate Key 1 (2, 3)

  Specify whether the duplication of alternate key attributes is unacceptable (1) or acceptable (2).

- Position and Length of Alternate Key 1 (2, 3)

  Specify the position and length of each alternate key just as for the primary key.

- End Code

  Specify 0 to mark the end of the file structure information.

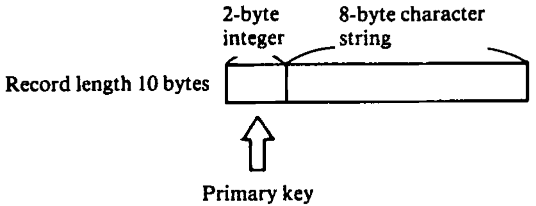
Specify an integer-type variable in <Return Code>. The return code indicating the result of execution of the ISAM OPEN statement is assigned to the variable specified here. When the files are opened correctly, 0 is assigned. <Return Code> must be specified in the operand of every ISAM instruction. When 0 is assigned to this variable, it means that the ISAM instruction was executed correctly. Return codes will not be explained later. Refer to "4.8 Return Codes" for the meanings of the return codes.

When a file on the disk will be opened with the ISAM OPEN statement, the file structure specified with this statement is checked to see if it is the same as the structure of the file on the disk. An error occurs if they are not the same. If fewer alternate keys are specified than the number of alternate keys of the file on the disk, a warning return code is returned and the processing continues.

When a file is opened in the input or update mode, the record that has the lowest primary key value in the file becomes the current record, and the primary key becomes the current key.

[Ex. 4.4.1-1]

A file "DFILE. ISM" consisting of the following records is created on the disk in drive A:




```
10  INTEGER ID, PARM, STAT
20  DIM BUF$1(10), PARM(5)
30  PARM(1)=2: REM OUTPUT MODE
40  PARM(2)=10: REM RECORD LENGTH=10 BYTES
50  PARM(3)=1: REM P KEY POSITION=1ST BYTE
60  PARM(4)=2: REM P KEY LENGTH=2 BYTES
70  PARM(5)=0: REM END CODE
80  ISAM OPEN (ID, "DFILE.ISM", PARM(*), STAT)
 .
 .
 .
```

In this example, the INTEGER and DIM statements on lines 10 and 20 define the integer-type variables required to execute the ISAM OPEN statment.

ID is the variable to which the file number of the pair of files opened is assigned. STAT is the variable to which the return code is assigned.

Array variables PARM (1) ~ PARM (5) are integer-type array variables that specify the structure of the files that will be opened. The values of file structure information are assigned to the variables by the LET statements (with the keyword omitted) on lines 30 ~ 70. The number of the array elements is 5 because alternate keys are not specified. 2 is assigned to variable PARM(1) to specify the output mode, and 10 is assigned to variable PARM(2) as the length of the record. Because the primary key position is at the 1st byte from the beginning, 1 is assigned to PARM(3) as the key position and 2 is assigned to PARM(4) as the key length. 0 is assigned to PARM(5) as the end code.

The ISAM OPEN statement on line 80 actually opens the files for indexed access. With the execution of this ISAM OPEN statement, data file "DFILE. ISM" and index file "DFILE.IDX" are opened on the disk in drive A. After the files are opened correctly, 0 is assigned to variable STAT, and the file number (1 ~ 6) is assigned to variable ID. After the execution of the ISAM OPEN statement, processing to the files is performed by specifying the file number assigned to ID.

4.4.2 ISAM CLOSE Statement (ISAM Close)

Function

This statement closes a pair of data files and index files.

Format

ISAM CLOSE(<File No.>, <Return Code>)

Explanation

The ISAM CLOSE statement closes the pair of files specified by the file number.

The open data file and index file must be closed with this statement before the program ends. Ending the program without closing the files may cause data irregularities when additional writing, modification, or other processing is performed.

[Ex. 4.4.2-1]

The file opened in Ex. 4.4.1-1 is closed.

```
10  INTEGER ID,PARM,STAT
 :
 :
80  ISAM OPEN(ID,"DFILE.ISM",PARM(*),STAT)
 :
 :
500 ISAM CLOSE(ID,STAT)
510 IF STAT=0 THEN PRINT "NORMAL CLOSE":GOTO 530
520 PRINT "CLOSE ERROR";STAT
530
```

The IF statement of line 510 confirms that the file has been closed correctly.

4.5 ISAM Data Write Instructions

4.5.1 ISAM PACK Statement (ISAM Pack)

Function

This statement assigns data to the record buffer.

Format

```
ISAM PACK(<Buffer>, <Expression>, ······ , <Expression> ,
                                           <Return Code>)
```

Explanation

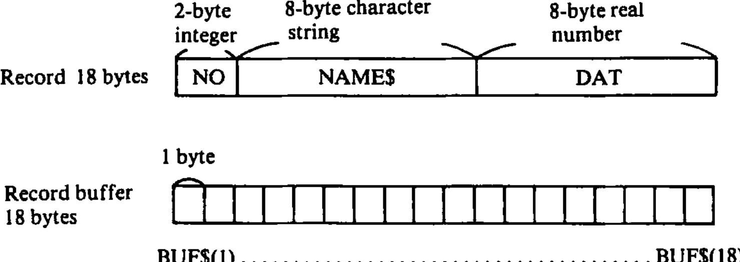
The ISAM PACK statement assigns data to the record buffer. Assigning data to the record buffer is always performed using this statement.

The part of a record buffer to which data will be assigned can be specified. Data can also be rewritten in a part of the record buffer.

Specify the string array variable, and its subscript, defined as a record buffer, in <Buffer>. The array variable specified here indicates the position in the record buffer where data assignment will start. For example, if BUF$(1) ~ BUF$(15) are defined as a record buffer, specify BUF$(3) to start data assignment at the third byte of the record buffer.

Specify the data that will be assigned to the record buffer in <Expression>. More than one block of data can be specified, but the data specified must conform to the data structure of the record (the length of each block of data).

The relationship between the data assigned to a record buffer and the specification of a record buffer is shown on the next page.



- When one record of data is assigned to the record buffer:

```
ISAM PACK(BUF$(1),NO,NAME$,DAT,STAT)
```

(BUF$(1): From head of record buffer; NO,NAME$,DAT: Data)

- When only the data of NAME$ is assigned to the record buffer:

```
ISAM PACK(BUF$(3),NAME$,STAT)
```

(BUF$(3): From 3rd byte of record buffer; NAME$: Data)

Each specified length of data is assigned to the record buffer starting from the specified position. Data assignment can start at any position in the record buffer. Be sure that the data assigned does not exceed the record length.

Note

When string data or a value is specified directly in <Expression> of the operand of the ISAM PACK statement, the legth of data is:

```
ISAM PACK(BUF$(3),"ABC",STAT)
```

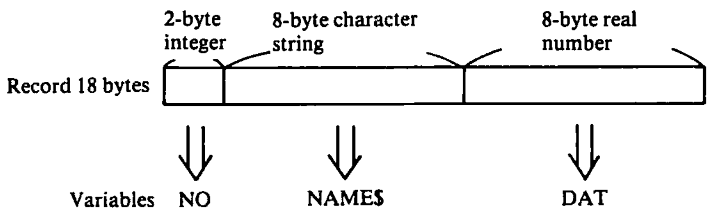
("ABC": 3 bytes (number of characters = number of bytes))

```
ISAM PACK(BUF$(11),342,STAT)
```

(342: 8 bytes (Numeric value is 8 bytes))

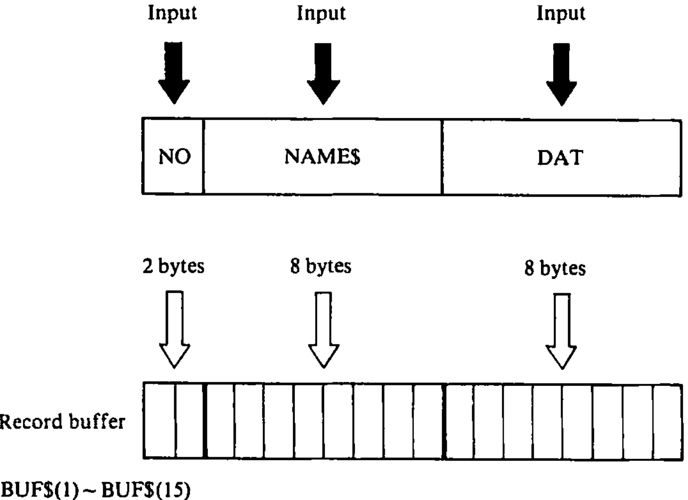
[Ex. 4.5.1-1]

The data of a record is assigned to the record buffer:




```
10  INTEGER NO,ID,PARM,STAT
20  DIM BUF$1(18),PARM(5)
 :
80  ISAM OPEN(ID,"DFILE.ISM",PARM(*),STAT)
 :
200 INPUT MSG("NO.?")NO:PRINT
210 INPUT MSG("NAME?")NAME$:PRINT
220 INPUT MSG("DATA?")DAT:PRINT
230 ISAM PACK(BUF$(1),NO,NAME$,DAT,STAT)
 :
```

Line 20 defines a record buffer length of 18 bytes with 1-byte string-type array variables BUF$(1) ~ BUF$(18). After the file is opened on line 80, the data input on lines 200 ~ 220 is assigned to the record buffer using the ISAM PACK statement on line 230.



4.5.2 ISAM WRITE Statement (ISAM Write)

Function

This statement writes the contents of an additional record to a file.

Format

```
ISAM WRITE(<File No.>, <Buffer>, <Return Code>)
```

Explanation

The ISAM WRITE statement writes an additional record to a file opened in the output or update mode.

Specify the file number, defined when opening the file, to which additional writing will be performed in <File No.>. The file to which additional writing will be performed must be a file opened in the output mode (2) or update mode (3). An error occurs if this statement is executed on a file opened in the mode.

Specify all of the elements of the string-type array variables defined as the record buffer with an asterisk (*) in <Buffer>.

Before the execution of this statement, the data of the record that will be written must be assigned to the record buffer using the ISAM PACK statement.

This statement checks for the duplication of the key values of the record that will be written and the key value of all records in the file before the additional writing of the record. Because the duplication of primary key values is unacceptable, if the value of the primary key of the record that will be written is the same as the value of the primary key of any other records in the file, return code 2 is returned and the additional writing of the record is not performed.

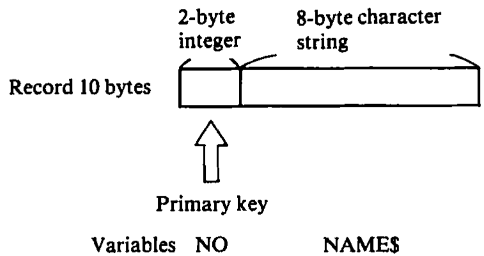
This also applies to the alternate keys for which duplication is unacceptable. If the values of the keys for which duplication is specified as acceptable are duplicated, warning return code 3 is returned and the additional writing of the record is performed.

Note

The current record and the current key are reset after execution of the ISAM WRITE statement.

[Ex. 4.5.2-1]

File "FILE1.ISM", consisting of records with the following data structure, is created on the disk in drive A.




```
 10 INTEGER NO,ID,PARM,STAT
 20 DIM BUF$1(10),PARM(5)
 30 PARM(1)=2:REM OUTPUT MODE
 40 PARM(2)=10:REM RECORD LENGTH = 10 BYTES
 50 PARM(3)=1:REM P KEY POSITION = 1ST BYTE
 60 PARM(4)=2:REM P KEY LENGTH = 2 BYTES
 70 PARM(5)=0:REM END CODE
 80 ISAM OPEN(ID,"FILE1.ISM",PARM(*),STAT)
 90 INPUT MSG("NO.=")NO:PRINT
100 IF NO=0 GOTO [END]
110 IF NO<0 GOTO 90
120 INPUT MSG("NAME=")NAME$:PRINT
130 ISAM PACK(BUF$(1),NO,NAME$,STAT)
140 ISAM WRITE(ID,BUF$(*),STAT)
150 IF STAT=-2 THEN PRINT "DUP. ERROR!":GOTO 90
160 GOTO 90
170 [END] ISAM CLOSE (ID, STAT)
 :
```

A file in which one record is 10 bytes (integer data: 2 bytes; character data: 8 bytes) is created for indexed access.

The LET statements on lines 30 ~ 70 assign the file structure information to array variables PARM (1) ~ PARM (5). The output mode (2) is selected as the open mode to create a new file.

The data input routine is on lines 90 ~ 120. The data of the record that will be written is input through the keyboard.

After data is input to the variables NO and NAME$, the data is assigned to the record buffer with the ISAM PACK statement on line 130. In this case, the data of the entire record is assigned to the record buffer so BUF$(1) is specified in the operand of the ISAM PACK statement to assign the data from the beginning of the record buffer.

The contents of the record buffer are written to the file using the ISAM WRITE statement on line 140. If the duplication of key values occurs between the data of the record that will be written and the data of the record in the file, the message "DUP. ERROR" is output by the IF statement on line 150 and writing is not performed.

4.5.3 ISAM REWRITE Statement (ISAM Rewrite)

Function

This statement rewrites the record data in a file.

Format

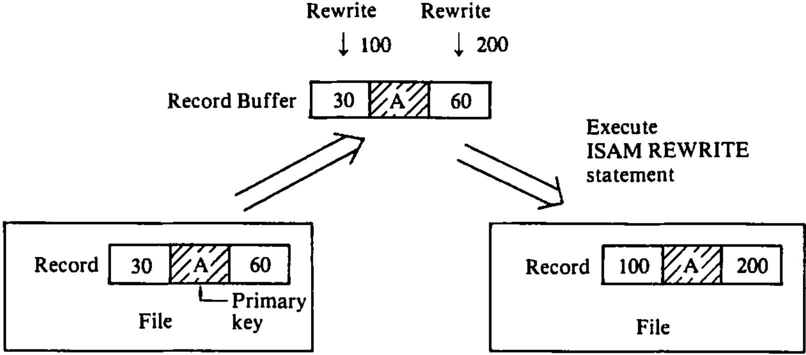
ISAM REWRITE(<File No.>, <Buffer>, <Return Code>)

Explanation

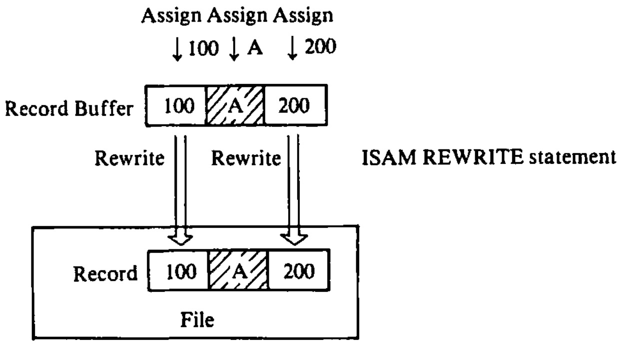
The ISAM REWRITE statement updates the record in the file whose primary key value is the same as the primary key value specified in the record buffer.

Specify the number of the file that will be rewritten in <File No.>. Specify the variable defined in the record buffer in <Buffer>.

For example, if data is read from a file to the record buffer and this statement is executed after the modification of a part other than the primary key of the data, that part of the record from which the data was read, is rewritten.



The same result is obtained by assigning the same value as the primary key value of the record that will be rewritten to the record buffer, without reading the data from the file.



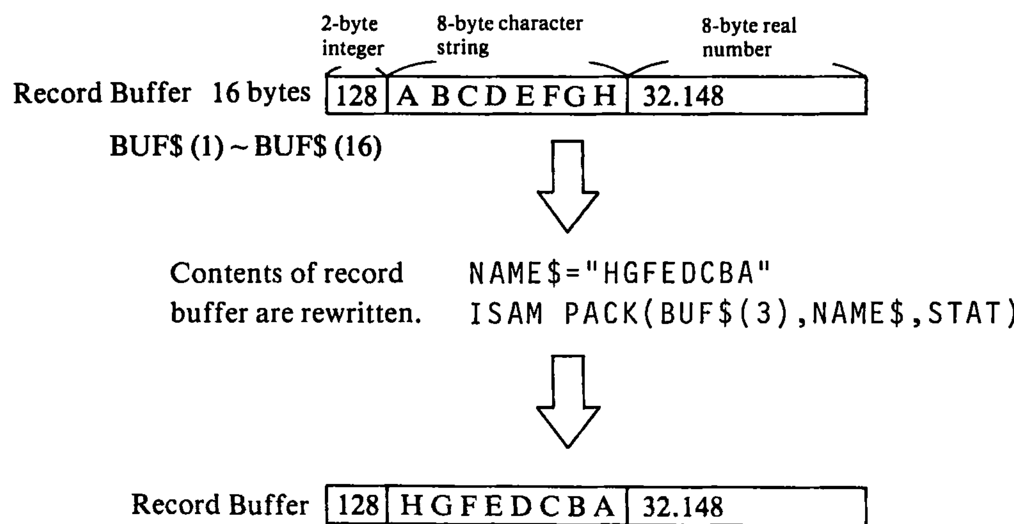
In this case, unless the record having the primary key value specified in the record buffer was written in the file, the record is not rewritten and return code 2 is returned.

When the modification of an alternate key value is specified, the newly specified alternate key value is checked to see if it duplicates the key value of record in the file. If an alternate key value is duplicated when duplication is specified as unacceptable, return code 2 is returned and the record is not rewritten. If the duplication of an alternate key value is specified as acceptable, return code 3 is returned and the record is rewritten.

Note

Only the part rewritten with the ISAM PACK statement is changed.

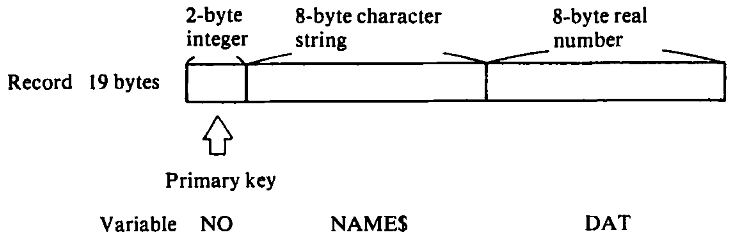
[Ex.]



The current record and the current key are reset when the ISAM REWRITE statement is executed.

[Ex. 4.5.3-1)

Rewrite data in a file written with the following record.



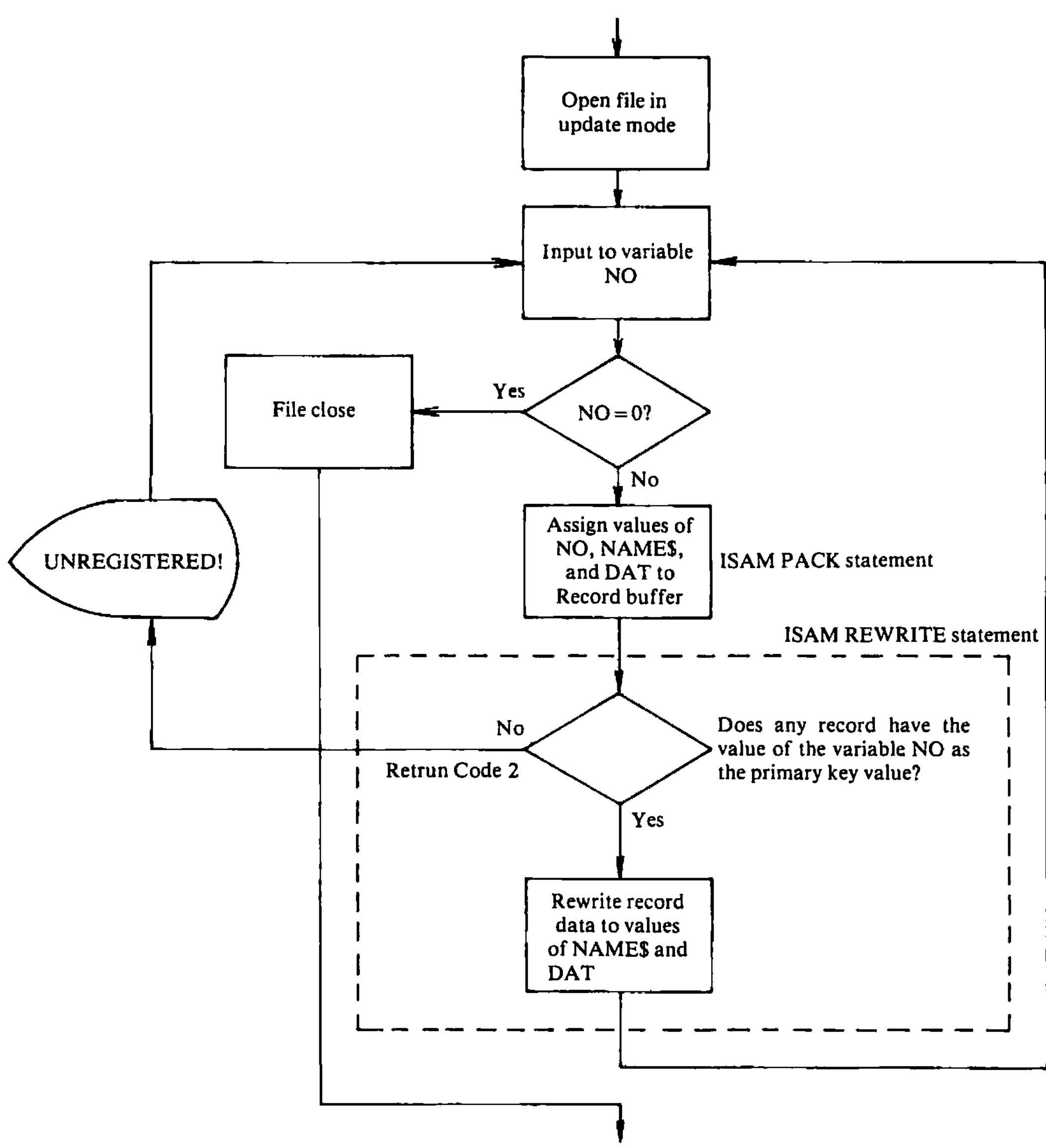

```
10  INTEGER NO,ID,PARM,STAT
20  DIM BUF$1(18),PARM(5)
30  PARM(1)=3:REM UPDATE MODE
40  PARM(2)=18:REM RECORD LENGTH = 18 BYTES
50  PARM(3)=1:REM P KEY POSITION=1ST BYTE
60  PARM(4)=2:REM P KEY LENGTH=2 BYTES
70  PARM(5)=0:REM END CODE
80  ISAM OPEN(ID,"DFILE.ISM",PARM(*),STAT)
90  INPUT MSG("NO.")NO:PRINT
100 IF NO=0 GOTO [END]
110 INPUT MSG("NAME=")NAME$:PRINT
120 INPUT MSG("DATA=")DAT:PRINT
130 ISAM PACK(BUF$(1),NO,NAME$,DAT,STAT)
140 ISAM REWRITE(ID,BUF(*),STAT)
150 IF STAT=-2 THEN PRINT UNREGISTERED!":GOTO 90
160 GOTO 90
170 [END]ISAM CLOSE(ID, STAT)
 :
```

The data of the record in the file is rewritten according to the primary key value, so the file is opened in the update mode (3).

The data is assigned to the record buffer by the ISAM PACK statement on NAME$ and DAT with the INPUT statements on lines 110 and 120.

The data is assigned to the record buffer by the ISAM PACK statement of line 130, and the data of the file is rewritten with the ISAM REWRITE statement on line 140. If the record having the primary key value specified (input to the variable NO) is not in the file (return code 2), the data of the record is not rewritten and the message "UNREGISTERED!" is displayed, requiring reinput of data.

This processing is shown below.



4.6 ISAM Data Read Instructions

4.6.1 ISAM UNPACK Statement (ISAM Unpack)

Function

This statement reads data from a record buffer and assigns it to a variable.

Format

```
ISAM UNPACK(<Buffer>, <Variable>, ···, <Return Code>)
```

Explanation

The ISAM UNPACK statement is the opposite of the ISAM PACK statement. It reads data from any part of the record buffer and assigns it to a variable.

Specify the elements of the operand as for the ISAM PACK statement.

The length of data that will be read from the record buffer depends on the length of a variable specified in the operand. Be careful not to specify the reading of data that exceeds the length of the record buffer.

An example of a program using this statement is shown in the explanations of the ISAM RREAD statement and ISAM SREAD Statement.

4.6.2 ISAM RREAD Statement (ISAM Random Read)

Function

This statement reads record data by specifying a key value. (Indexed random read).

Format

```
ISAM RREAD(<File No.>, <Buffer>, <Return Code>[, <Key Type>])
```

Explanation

This statement searches a file for a record having a specified key value and reads the data from that record into the record buffer.

A key value is specified by assigning the value to the key part of the record buffer before this statement is executed.

Specify the file number defined when opening the file from which data will be read in <File No.>. The file must be opened in the input or update mode.

For <Buffer>, specify all of the elements of the string-type array variables defined as the record buffer with an asterisk (*).

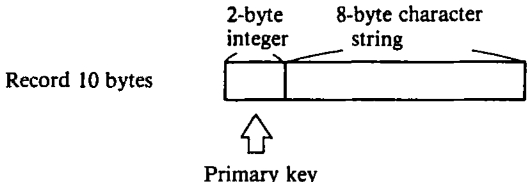
Specify an integer-type variable in <Key Type>. This variable must be selected from the integer values 1～4, which indicate the key type serving as an index for record retrieval and assigned before the execution of this statement. 1 indicates the primary key, 2 indicates alternate key 1, 3 indicates alternate key 2 and 4 indicates alternate key 3. If the specification is omitted, 1 (the prima rykey) is automatically specified.

After this statement is executed, the current key becomes the type of key specified with this statement, and the next key in the key order to the record from which data was read becomes the current record.

When more than one record has the specified key value (i.e. when an alternate key accepting duplication is specified), data is read from the record written to the file first. In this case, the current record after the execution of this statement, is the record written to the file second.

When there is no record having the specified key value, reutrn code −2 is returned and data reading is not performed.

The indexed random read processing with this statement is shown below.



① Key value is assigned to record buffer.

135

NO = 135

ISAM PACK (BUF$ (1), NO, STAT)

Record Buffer | 135 |

② ISAM RREAD statement is executed.

Record Buffer | 135 |

Index file

Primary key information

No corresponding record

Return code −2

Corresponding record is found

Data file

③ Data is read from corresponding record into record buffer.

Pointer

62	S T U V W X Y Z
135	A B C D E F G H
205	O P Q R S T U V

Record Buffer | 135 | A B C D E F G H |

⑤ Contents of record buffer are read.

④ Pointer is moved.

| 62 | S T U V W X Y Z |
| 135 | A B C D E F G H |

Pointer (current record)

| 205 | O P Q R S T U V |

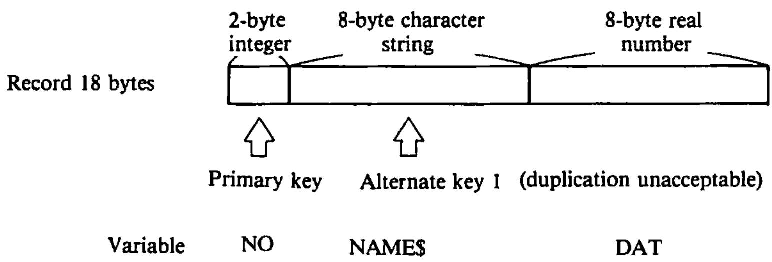
↑ Current key

ISAM UNPACK (BUF$ (3), NAME$, STAT)

Variable NAME$ | A B C D E F G H |

[Ex. 4.6.2-1]

The key value of alternate key 1 in a record with the following data is specified, and the data is read from the record in the file:




```
10  INTEGER NO,ID,PARM,STAT,KEYNO
20  DIM BUF$1(18),PARM(8)
30  PARM(1)=1:REM INPUT MODE
40  PARM(2)=18:REM RECORD LENGTH = 18 BYTES
50  PARM(3)=1:REM P KEY POSITION = 1ST BYTE
60  PARM(4)=2:REM P KEY LENGTH = 2 BYTES
70  PARM(5)=1:REM A1 KEY DUPLICATION UNACCEPTABLE
80  PARM(6)=3:REM A1 KEY POSITION = 3RD BYTE
90  PARM(7)=8:REM A1 KEY LENGTH = 8 BYTES
100 ISAM OPEN(ID,"FILE3.ISM",PARM(*),STAT)
110 INPUT MSG("NAME?")NAME$:PRINT
120 IF NAME$="E" GOTO [END]
130 ISAM PACK(BUF$(3),NAME$,STAT)
140 KEYNO=2:REM KEY TYPE = ALTERNATE KEY 1
150 ISAM RREAD(ID,BUF$(*),STAT,KEYNO)
160 IF STAT=-2 THEN PRINT "NAME UNREGISTERED!":GOTO 110
170 ISAM UNPACK(BUF$(1),NO,NAME$,DAT,STAT)
180 PRINT "NO.=";NO
190 PRINT "DATA=";DAT
200 GOTO 110
210 [END]ISAM CLOSE(ID, STAT)
 :
```

In this example, the value of alternate key 1, whose duplication is specified as unacceptable, is specified, and data is read from the record that has the specified key value. The file is opened in the input mode.

The INPUT statement on line 110 inputs the data (key value) serving as an index for record retrieval to the variable NAME$.

The ISAM PACK statement on line 130 assigns the key value input to NAME$ to the record buffer. Because data is assigned to only the alternate key 1 part of the record buffer, BUF$ (3) (from the third byte of the record buffer) is specified in the operand.

After the key value is assigned to the record buffer, the ISAM RREAD statement on line 150 reads data from the file. The LET statement on line 140 specifies the key type as alternate key 1.

If the record specified with the ISAM RREAD statement on line 150 is not in the file, return code −2 is returned. After the message "NAM EUNREGISTERED!" is displayed, the system again enters the input status.

The ISAM UNPACK statement on line 170 assigns the contents of the record buffer to the variable.

4.6.3 ISAM START Statement (ISAM Start)

Function

This statement defines the reading start record for the sequential reading of record data according to the key order. (Indexed Sequential Read).

Format

```
ISAM START(<File No.>, <Buffer>, <Start Code>, <Return Code>
                                                [, <Key No.>])
```

Explanation

The ISAM START statement defines the first record where the reading data will start when indexed sequential read is performed with the ISAM SREAD statements described later. Actually the pointer moves according to the conditions specified with this statement, and the current record and the current key are set.

As with the ISAM RREAD statement, the conditions for setting the record to start reading data are specified by assigning a specified key value to the record buffer in advance and specifying the variable of the record buffer and the key type (<Key No.>) of the record in the operand of the ISAM SREAD statement. With this statement, however, it is possible to specify a comparison between the key value of the record that will be searched and the specified value. (With the ISAM RREAD statement, the record that has the same key value as the specified value is searched.)

For <Start Code>, specify an integer-type variable, and assign 0, 1, or 2 to it according to the conditions of key values comparison in advance. Specify the start code as follows:

0..... Specified key value = Record key value
- The record that has the same key value as the specified key value is the current record

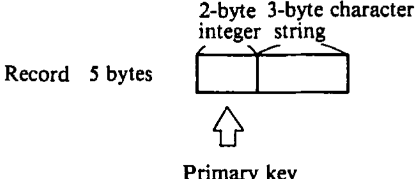
1..... Specified key value ≦ Record key value
- The record that has the same key value as the specified key value is the current record. When there is no record with the same key value, the record that has the next key value in the key order is the current record.

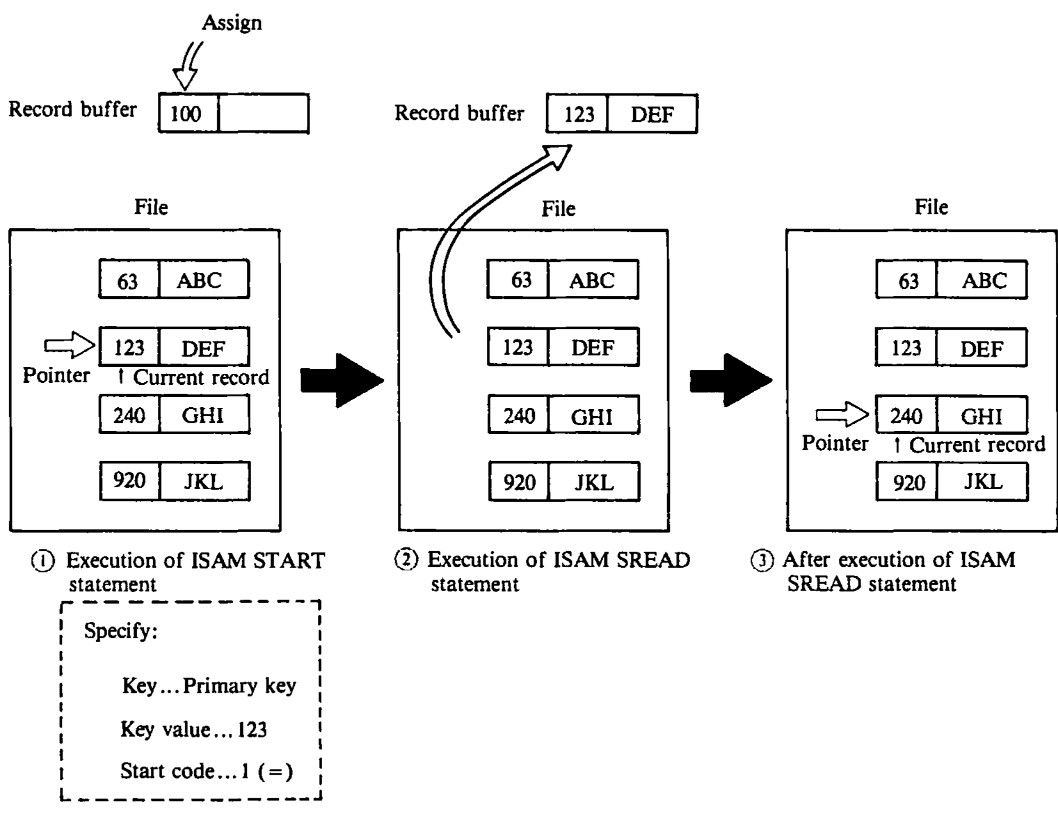
2..... Specified key value < Record key value

- The record that has the first key value larger than the specified key value is the current record.

When there is more than one record that has the same key value corresponding to the specified conditions (when an alternate key accepting duplication is specified), the record written to the file first is the current record. If there is no corresponding record, return code −2 is returned and the pointer is not defined.

The movement of the pointer by the execution of this statement and the ISAM SREAD statement is shown below.





① Execution of ISAM START statement

② Execution of ISAM SREAD statement

③ After execution of ISAM SREAD statement

An example of a program using this statement is shown in the explanation of the ISAM SREAD statement.

4.6.4 ISAM SREAD Statement (ISAM Sequential Read)

Function

This statement reads data in sequence from the records of a file according to the key order. (Indexed Sequential Read.)

Format

ISAM SREAD(<File No.>, <Buffer>, <Return Code>)

Explanation

The ISAM SREAD statement reads one record of data from the current record of the file into the record buffer and moves the pointer to the next record in the key order of the current key. Data in records are read in sequence according to the key order of the current key (indexed sequential read) with the repeated execution of this statement.

Specify the file number defined when opening the file from which data will be read in <File No.>. The file must be opened in the input or update mode.

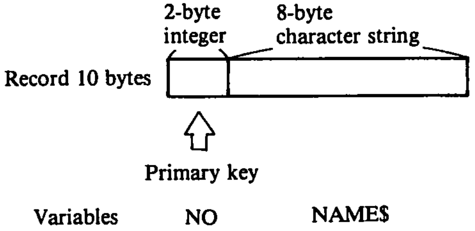
For <Buffer>, specify all elements of string-type array variables defined as the record buffer with an asterisk (*).

The ISAM START statement must be executed to set the current record and the current key before the start of indexed sequential access by this statement. But when the primary key is used as the current key and indexed sequential read starts with the record with the lowest key value, the ISAM START statement need not be executed because the current record and the current key are set automatically just after execution of the ISAM OPEN statement. Remember that the current record and the current key change when the ISAM RREAD statement is executed.

When the current key is an alternate key whose duplication is specified as acceptable and there is more than one record that has the same key value, the records are read in the order in which they were written to the file.

[Ex. 4.6.4-11]

Assuming that the records consist of the following data, data is read in the key order from the records whose primary key values are between 100 and 200:




```
10  INTEGER NO,ID,PARM,STAT,SCODE
20  DIM BUF$1(10),PARM(5)
30  PARM(1)=1:REM INPUT MODE
40  PARM(2)=10:REM RECORD LENGTH: 10 BYTES
50  PARM(3)=1:REM P KEY POSITION: 1ST BYTE
60  PARM(4)=2:REM P KEY POSITION: 2ND BYTE
70  PARM(5)=0:REM END CODE
80  ISAM OPEN(ID,"FILE3.ISM",PARM(*),STAT)
90  PRINT "CODE","NAME":PRINT
100 SCODE=1:NO=100:REM  100<=CODE
110 ISAM PACK(BUF$(1),NO,STAT)
120 ISAM START(ID,BUF$(*),SCODE,STAT)
130 ISAM SREAD(ID,BUF$(*),STAT)
140 IF STAT=-2 GOTO [END]
150 ISAM UNPACK(BUF$(1),NO,NAME$,STAT)
160 IF NO>200 GOTO [END]
170 PRINT NO,NAME$
180 GOTO 130
190 [END]ISAM CLOSE(ID, STAT)
 :
```

In this example, data is read from the records in the key order of primary key values and is displayed.

The file from which data is read is opened in the input mode.

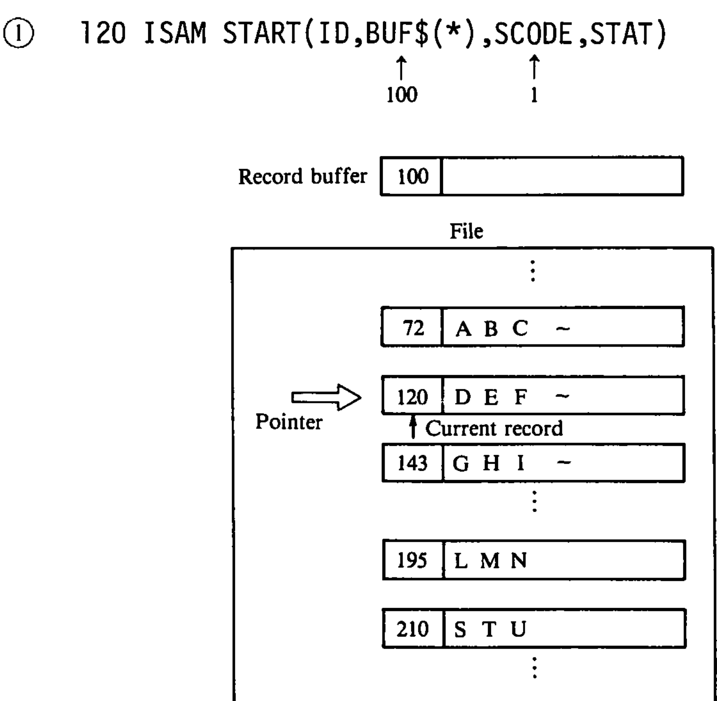
The two LET statements on line 100 specify the key value of the record where data reading will start. They assign 1 as the start code and 100 as the key value to each variable and specify the current record to the first record in the key order from the records whose primary key values are 100 or more.

The ISAM START statement on line 110 actually moves the pointer to the record mentioned before. At the same time, the primary key becomes the current key.

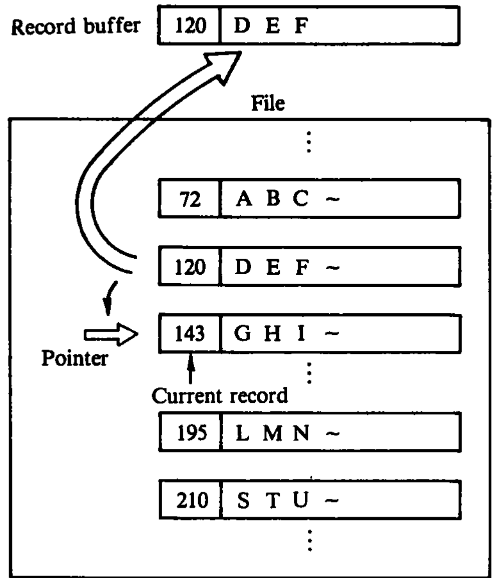
The indexed sequential read routine is on lines 120 ~ 180. It is a loop to repeat execution the required number of times. In this loop, the ISAM SREAD statement is executed repeatedly to read data from records according to the key order of the records' data.

When the primary key value of the record read exceeds 200, the loop ends and the file is closed. When there is not a record whose primary key value is larger than 200, data reading reaches the end of the file before the data that exceeds 200 is read. In this case, return code −2 is returned and the loop ends with the IF statement on line 140.

The processing for this program example is shown next.

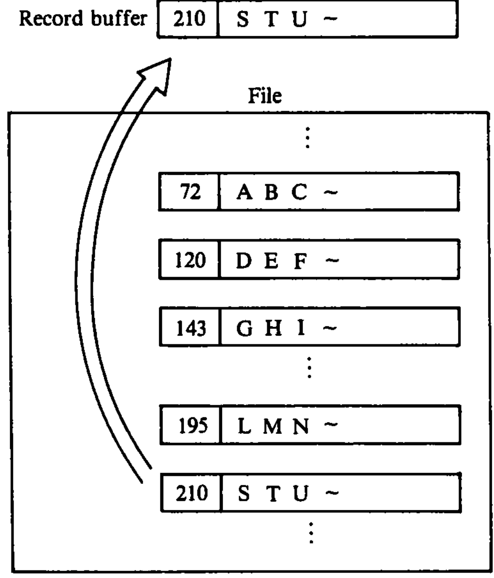


② 130 ISAM SREAD(ID,BUF$(*),STAT)



③ The ISAM SREAD statement is repeated.

④ 130 ISAM SREAD(ID,BUF$(*),STAT)



⑤ 160 IF NO>200 GOTO [END]

190 [END]ISAM CLOSE(ID,STAT)

4.7 Other ISAM Instructions

4.7.1 ISAM DELETE Statement (ISAM Delete)

Function

This statement deletes a specified record from a file.

Format

ISAM DELETE(<File No.>, <Buffer>, <Return Code>)

Explanation

The ISAM DELETE statement deletes the record that has the same primary key value as the one specified in the record buffer from an index file. Records in the data file are not deleted with this statement. But, because the key information in the index file is deleted, the record deleted with this statement cannot be read with indexed access. The area of the data record deleted with this statement is used for the additional writing of a record.

Specify the file number defined when opening the file from which a record will be deleted for <File No.>. The file must be opened in the output mode or update mode.

For <Buffer>, specify all of the elements of string-type array variables defined as the record buffer with an asterisk(*). Assign the primary key value of the record that will be deleted to the record buffer before this statement is executed.

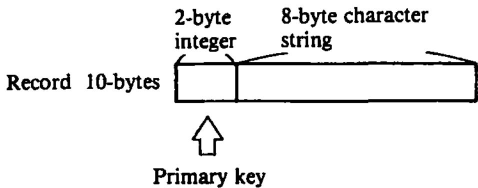
If the record specified is not in the file, return code −2 is returned.

Note

The current record and the current key are reset after execution of the ISAM DELETE statement.

[Ex. 4.7.1-1]

Delete all records with a primary key value of 100～200.




```
10  INTEGER NO,ID,PARM,STAT
 :
80  ISAM OPEN(ID,"FILE3.ISM",PARM(*),STAT)
90  FOR NO=100 TO 200
100 ISAM PACK(BUF$(1),NO,STAT)
110 ISAM DELETE(ID,BUF$(*),STAT)
120 IF STAT<>-2 THEN PRINT "DELETE";NO
130 NEXT NO
140 ISAM CLOSE(ID,STAT)
```

In this example, records whose primary key values are 100～200 are deleted.

The IF statement on line 120 checks the return code after execution of the ISAM DELETE statement and displays the primary key value of the record deleted.

4.7.2 ISAM SECUR Statement (ISAM Security)

Function

This statement specifies the period for writing into a file from the ISAM system buffer.

Format

```
ISAM SECUR(<Switch>, <File Information>, <Return Code>)
```

Explanation

When data is written to a file with indexed access, the data is temporarily stored in a memory area called the ISAM system buffer where it is defined automatically by ISAM before it is written to the data file and index file.

When the program ends abnormally during file updating, the data written to the file may be wrong. It number of data written to a file correctly may differ depending on the data writing period from the ISAM system buffer to the file.

The ISAM SECUR statement specifies the period for writing the contents of the ISAM system buffer to the file.

This specification determines the processing speed and the security of data.

In <Switch>, specify an integer-type variable. Assign 1, 0, or −1 to this variable indicating the period of writing the contents of the ISAM system buffer. These values have the following meanings:

1 ..... Each time the statement for writing data to a file is executed, the contents of the ISAM system buffer are written to the file. Although the processing speed is slow, the highest degree of file security is guaranteed.

0 ..... When the contents of the ISAM system buffer reaches capacity, the contents are written to the file. The disk is always checked to ensure that there is suffucient area to write the contents of the ISAM system buffer. The processing speed is higher than in 1, and an medium degree of file security is guaranteed. This method is adopted when the ISAM SECUR statement is not executed.

−1 ..... This method is the same as 0. But the disk area availability is not checked until the writing of the contents of the ISAM buffer is performed. Although the processing speed is the highest, sometime the contents of the ISAM system buffer cannot be written to the disk.

In <File Information>, specify the elements of the integer-type array variables defined for the file information with an asterisk(*)

Specify the integer-type array variables for <File Information>. The array rariable must consist of 6 array elements. The information of each file is assigned to each of the variables as follows when this statement is executed. If 0 is assigned, the file is normal. If the file is abnormal, 1 is assigned.

Ex: IFINF (1) ~ (6)

File No. 1	File No. 2	File No. 3	File No. 4	File No. 5	File No. 6
⇓	⇓	⇓	⇓	⇓	⇓
IFINF(1)	IFINF(2)	IFINF(3)	IFINF(4)	IFINF(5)	IFINF(6)

[Ex. 4.7.2-1]

ISAM processing is performed with 1 (the highest degree of file security) specified in switch.

```
10  INTEGER ID,PARM,STAT,SW,IFINF
20  DIM BUF$1(10),PARM(5),IFINF(6)
 :
 :
400 SW=1
410 ISAM SECUR(SW,IFINF(*),STAT)
 :
```

4.8 Return Codes

The values of the return codes returned by ISAM statements have the following meanings:

<Normal>

0 ..... Normal end

<Errors>

1, 3, 4, 6 and 10 ..... The index file is wrong.

2 ..... The disk is full.

5 ..... The directory area of the disk has overflowed.

7 ..... The specified file is not on the disk.

8 ..... The disk or file has a read-only attribute.

9 ..... An attempt is made to open the same file twice.

11 ..... The number of records in the file has exceeded the limitation.

12 ..... The EOD record is wrong.

13 ..... There is no EOD record.

20 ..... The parameter attribute is wrong.

21 ..... An attempt is made to open more than six pairs of files at the same time.

32 ..... The file name specification is wrong.

23 ..... The parameter value is wrong.

24 ..... The record length is wrong.

26 ..... An attempt is made to execute an instruction that cannot be executed in the specified open mode.

27 ..... The key specification is wrong.

28 ..... When an attempt is made to open files in the update mode, either the data file or the index file is not on the disk.

29 ..... The number of a file that is not opened is specified.

30 ..... The contents of the index file are wrong.

31 ..... Error in the ISAM SECUR statement.

32 ..... The index file size has exceeded the limitation.

<Warnings>

−1 ..... An attempt is made to read data beyond the end of data.

−2 ..... The key specification is wrong.

−3 ..... The key value is duplicated.

−4 ..... Because the index file is wrong, the alternate key position is wrong.

−5 ..... The number of keys specified with file structure information is greater than the number of the keys of the records in the file opened.

−6 ..... The record deleted is wrong.

Warning codes −1 and −2 indicate the abnormal execution of the ISAM instructions and record data is not read or written.

When −4 or −6 is returned, the statement is executed but the result is not guaranteed because the data file or index file is abnormal.

Note

When the integer-type variable for <Return Code> is not specified in the operand or when the specified variable is not of the integer-type, a return code is not returned.

Caution **Disk Overflow**

The disk may overflow during ISAM processing. This may occur in following cases:

When writing to a data file .... ISAM WRITE statement execution
When writing to an index file ..... ISAM WRITE, REWRITE, DELETE, CLOSE statement executions

In these cases, return code 2 is returned and the file may become invalid for ISAM. So use a disk which has sufficient free area for ISAM to prevent the overflow. This error may occur even if the ISAM SECUR statement, which specifies switch 1, is executed.

If the file becomes invalid because of this error, reorganize the file using the ISGEN utility and reactivate the file for ISAM. Refer to "4. 9. 1 ISGEN Utility" for details of this recovery operation.

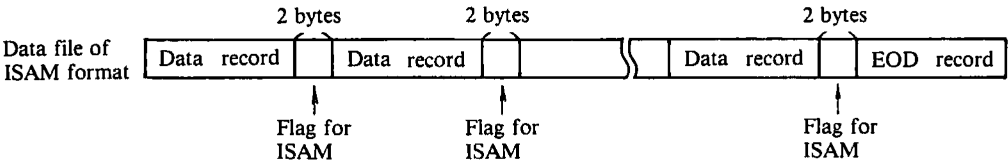
4.9 ISAM Utility Programs

The ISGEN utility and IDXINF utility are utility programs that can be executed in the OS mode. This part explains the functions and the procedures for using these two utility programs.

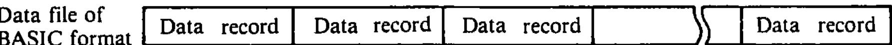
4.9.1 ISGEN Utility

The ISGEN utility is a utility program to generate an index file from a data file. It can be used to regenerate an index file from a data file that has lost an index file for some reason, or to create a pair of files for indexed access from a data file created using the PUT statement.

In data files used in ISAM, a 2-byte flag for ISAM is attached to the end of each record. The flag is "$0D_H0A_H$" if the record is active (not deleted) and "00_H00_H" if the record is deleted. An EOD record containing information for ISAM is also attached to the end of data. This data file for ISAM is called a data file of the ISAM format.



In contrast, the following file created with the PUT statement is called a data file of the BASIC format:



The functions of the ISGEN utility are next described in detail for files of these two different formats.

One of the functions of this utility is to regenerate an index file according to the data of an ISAM format data file that has lost a corresponding index file, thus reenabling indexed access.

It is also possible to reorganize files for ISAM according to the contents of the data file or index file which have become unusable for ISAM because of a disk overflow.

The other function is to generate a pair of files for indexed access according to a data file of the BASIC format created with the PUT statement.

This utility is used in the interactive operation.

The starting method and operation are shown below.

«Display»	«Operation and Explanation»
A>_	• Make sure that the OS mode is set.
	[I][S][G][E][N] [↵]
① *Source File Name?	
	<File Name> [↵] • Enter the name of the data file that serves as the source for generation of an index file.
② *Type of Data File (Reformat/Isam/Basic)?	
	[R][↵] or [I][↵] or [B][↵] • Enter I if the data file is of the ISAM format and B if it is of the BASIC format. If R is entered, the file reformatting (reorganization) is performed with processing as follows: a) The data file is reformatted by physically deleting the records which are deleted logically by ISAM DELETE statement. b) Reorganization of the files which have become unusable with disk overflow.

«Dipslay» «Operation and Explanation»

③ *Source Data Record Length?

<Record Length> ↵

- Enter the record length of one record in bytes. The value must be an integer within the range: 1 ~ 510. (When the data file has a ISAM format, omit the ISAM flag from the record length.)

Only when B or R is selected in ②.

④ *Data File Name to create?

<File Name> ↵

•Enter, together with the file type, the name of the data file of ISAM that will be created from the data file of BASIC format or from the incorrect data file of ISAM format.

Only when B is selected in ②.

⑤ *Creation Data Record Length?

<Record Length> ↵

- Enter the record length of the data file of ISAM format that will be created. If the record length entered here is less than the record length entered in ③, the excess records of the original file are ignored. If it is greater, 00_H is added to create file.

⑥ *Record Count?

BASIC format only

<Record Count> ↵

- Enter the number of records which will be converted for ISAM use. The default (↵ only) is that all records are converted.

⑦ *Primary Key Position?

<Primary Key Position> ↵

- Enter the position of the primary key.

«Dipslay» «Operation and Explanation»

⑧ * Primary Key Length?

<Primary Key Length> ↵

- Enter the length of the primary key.

⑨ * Transfer ratio 10∿90(%)[default=50%]?

<Transfer Ratio> ↵

- The transfer ratio is a value (an integer within the range: 10 ~ 90) that shows the size of the spare area of key information in the index file. The way the key information is split depends on this value.
  Enter an integer within the range: 10 ~ 90. For processing efficiency, enter 90 when the records of the original data file are arranged in the key order (from smallest to greatest values) and not many records will be added. Enter 50 when alot of records will be added or modified. Enter 10 when the records of the original file are arranged in the reverse key order (from greatest to smallest values). The default (↵ only) is that 50 is specified.

Repeat only when an alternate key is specified.

⑩ * Alternate key[0:No 1:Yes(No dup)2:Yes(dup)]?

[0] or [1] or [2] ↵

- Enter 1, 2, or 3 to specify whether an alternate key is set or not.
  If 0 is entered, an alternate key is not set. If 1 is entered an alternate key not accepting duplication is set.
  If 2 is entered, an alternate key accepting duplication is set. Steps ⑩ to ⑬ are repeated until 0 is entered here or three alternate keys are specified.

⑪ * Alternate Key Position?

<Key Position> ↵

- Enter the position of the alternate key.

«Dipslay» «Operation and Explanation»

⑫ *Alternate Key Length?

<Key Length> ↵
- Enter the length of the alternate key.

⑬ *Transfer ratio 10∿90(%)[default=50%]?

Repeat only when an alternate key is specified

<Transfer Ratio> ↵
- Enter the transfer ratio (10～90) as explained in 9 .

⑭ *ISGEN Start OK (No/Yes)?

[Y] (or [N]) ↵
- Make sure that the values entered are correct, then enter Y create the file. If N is entered, restart input at ①.

When there is already a file of the same name as the file that will be created on the disk, the message:

    * File Already Exists Delete (Yes/No)?

is displayed. If Y is entered, the file is deleted before the start of file creation. If N is entered, reinput of the file name is required.

During processing, the record number being processed is displayed.

The following error messages are displayed during the input of values or during processing:

«During the input of values»

Invalid File Name............The file name specification is wrong.

File doesn't exist..............The specified file is not on the disk.

Invaled Parameter............The value entered is wrong.

Read Only File................The file that will be deleted has a read-only attribute.

Invalid Key Data..............The specification of record length, key position, or key length is wrong.

«During processing»

Error: Code = <Return Code>

...This has the same meaning as the return code of ISAM instructions.

··Delete Record Found: Rec No = <Record No.>

...The record of <Record No.>in the original data file is a delete record. The record is also written as a delete record in the file that will be created.

··Invalid Record Flag: Rec No = <Record No.>

...The ISAM flag of the record having <Record No.> in the original file is wrong. The record is written as it is to the file that will be created.

··Invalid Key Found: Rec No = <Record No> Key No = <Key No> Ignored

The key value of <Record No.> is duplicated in <Key No.> even if duplication is not acceptable. The record is not written to the file that will be created.

··Duplicate Key Found: Rec No = <Record No.> Key No = <Key No.>

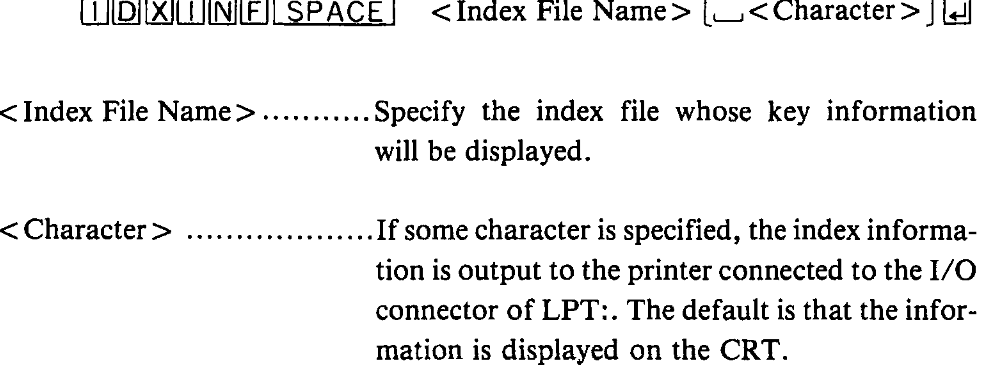
...The key value of <Record No.> is duplicated in <Key No.>when duplication is acceptable. The record is written to the file to be created.

The processing speed using ISGEN can be improved by rearranging, in advance, the records of the original data file in the key order. When more than one key is specified, rearrange the records on the basis of the key order of the greatest key length.

4.9.2 IDXINF Utility

This utility displays the key information of an index file to confirm the record length, record number, key position, key length, etc.

This utility is started with the following input in the OS mode.



[I][D][X][I][N][F][SPACE] <Index File Name> [␣<Character>] [↵]

<Index File Name> .... Specify the index file whose key information will be displayed.

<Character> .... If some character is specified, the index information is output to the printer connected to the I/O connector of LPT:. The default is that the information is displayed on the CRT.

The display of index information with this utility is shown next. The numbers ① to ⑥ explain the meanings of the messages.

[Example]

```
*** Index File Information Display Vn.nn ***

(1)  Data Record Length:   62 ··························①
(2)  Last Data Record:   1000 ··························②
(3)  Last Index Record:    35 ··························③
(4)  Free Data Record:    ·································④
           23       41
(5)  Free Index Record:  None ··························⑤
(6)  Key Informations:    ·································⑥
     Key no=1
           Position:     27
           Length:        5
           Root Rec.:     2
           Max Level:     1

**************** End of Information ***************
```

① Record length of data.
② Number of data record.
③ Number of index record. Not the number of data record.
④ Deleted records. Records 23 and 41 are deleted.
⑤ Free index record. Nil.
⑥ Key information. Only the primary key is set. The key position is the 27th byte from the beginning of the record and the key length is 5 bytes.

4.10 How To Calculate File Size

The method of calculating the size of ISAM data files and index files are shown here. The size of an index file is an approximate value based on the assumption that the transfer ratio is 50%.

According to CP/M-86 specifications, a file is written to a disk in 128-byte units and reserves disk area in 2K-byte units.

- Size of a data file (ISAM format)

  Size = (<Record length> + 2) × <Number of data record> (bytes)

- Size of an index file (Approximate value)

  Size = (<Key length> + 2) × <Number of data record>
× 1.1/0.7 (bytes)

ISAM Instruction Formats

① ISAM OPEN (<File No.>, <File Name>, <File Structure>, <Return Code>)

② ISAM CLOSE (<File No.>, <Return Code>)

③ ISAM PACK (<Buffer>, <Expression>,···, <Return Code>)

④ ISAM WRITE (<File No.>, <Buffer>, <Return Code>)

⑤ ISAM REWRITE (<File No.>, <Buffer>, <Return Code>)

⑥ ISAM UNPACK (<Buffer>, <Variable>, ..., <Return Code>)

⑦ ISAM RREAD (<File No.>, <Buffer>, <Return Code> [,<Key Type>])

⑧ ISAM START (<File No.>, <Buffer>, <Start Code>, <Return Code> [,<Key No.>])

⑨ ISAM SREAD (<File No.>, <Buffer>, <Return Code>)

⑩ ISAM DELETE (<File No.>, <Buffer>, <Return Code>)

⑪ ISAM SECUR (<Switch>, <File Information>, <Return Code>)

5. Graphic Functions

This section explains how to use the instructions for the various graphic functions of the AS-100 display through the Canon BASIC program.

5.1 Graphic Functions

The graphic functions enable the operator to draw various figures with the dots (640 x 400) on the display. The graphic functions use dots rather than characters as the unit of display.

The AS-100 graphic functions are supported by the CP/M-86 operating system and can be controlled by outputting defined codes to the display.

Refer to the CP/M-86 User's Manual for details of the display control codes.

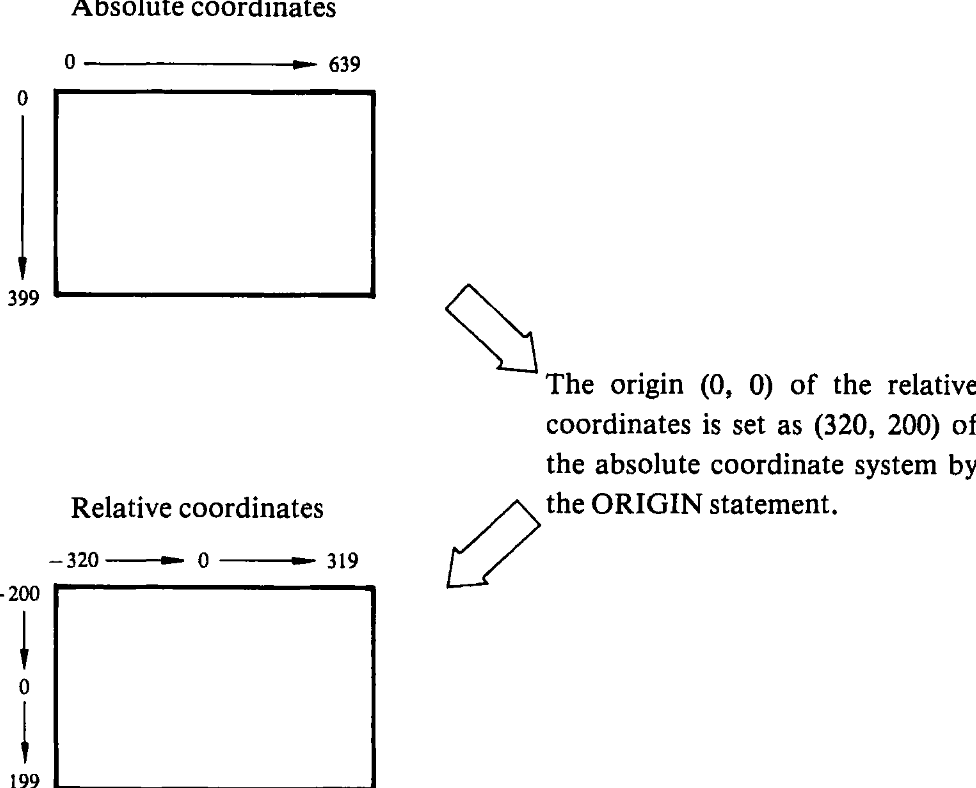
5.1.1 Coordinates

The location of a figure that will be drawn is specified with the coordinates of dots on the display.

The display screen has 0 ~ 639 coordinates on the X axis (horizontal) and 0 ~ 399 coordinates on the Y axis (vertical). They are coordinated with the dots on the display. These coordinates are called as absolute coordinates.

Relative coordinates are used for each instruction to facilitate coordinate specification in a program. Relative coordinates can be defined by the ORIGIN statement. When the ORIGIN statement is not executed, the absolute coordinates are identiacl to the relative coordinates.

The relationship between the absolute coordinates and the relative coordinates is as shown below.



Each of the coordinates is specified as (x, y). The figure will not be displayed when it is drawn at coordinates which are outside the display range. The valid range of x, y is $-32768 \sim 32767$ (absolute coordinates). Any numeric specification outside this range causes an error. A decimal within this range is automatically converted to an integer by truncating the decimal fraction.

The length of the figure that will be drawn is specified by the number of dots.

Note

The coordinates used for the graphic functions are not related to the coordinates of $x = 0 \sim 79$, $y = 0 \sim 24$ specified in %CURSOR of PRINT statement. Do not confuse the two.

5.1.2 Palette and Display Color Specification

Users can specify a display color (for color display) and a display mode (for monochrome display) with a palette. The palettes and display colors (display modes) are explained below.

AS-100 color display can display 27 colors. These 27 colors are numbered 0 ~ 26.

A palette is like a dish that stores one of these 27 colors. For example, a red circle can be drawn with the screen by specifying the red pallete.

The AS-100 color display has eight palettes, which are numbered 0 ~ 7. The palette used for each of the figures displayed is memorized so the display color can be changed by simply specifying the color of a palette. For example, assume that the red circle was drawn with palette No. 2. The red circle can be changed to white by changing the definition of palette 2 to ''white''.

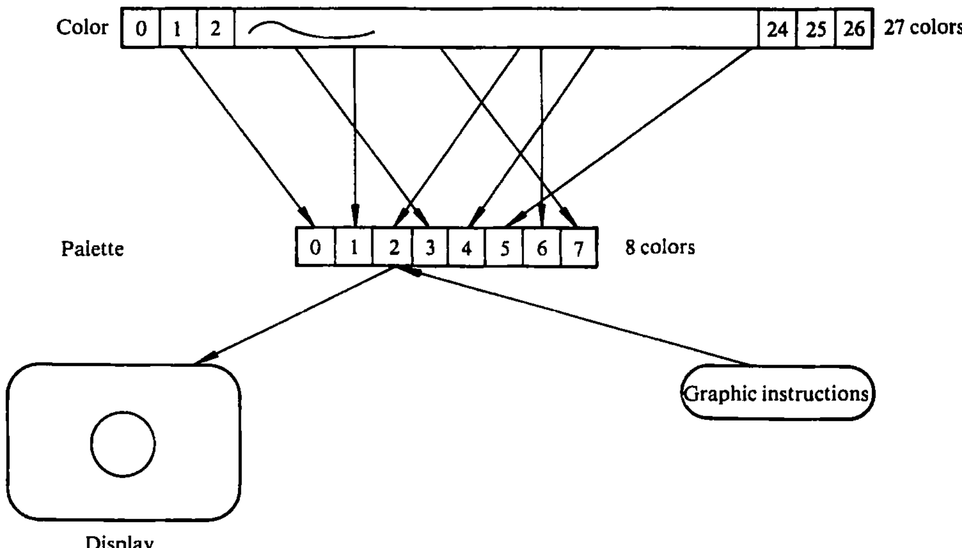
The same principle applies to the monochrome display.

The monochrome display supports 1-VRAM (Video RAM is the memory area for the data image displayed.) specification and 2-VRAM specifications. Like the color display, the monochrome display has a palette and color numbers. However, these color numbers are actually display mode numbers.

The 1-VRAM specifications of the monochrome display have two palettes and two color numbers (No-color = 0 and Colored = 1). The 2-VRAM specifications have four palettes and five color numbers (No-color, Standard Brightness, High Brightness, Blinking Standard Brightness, and Blinking High Brightness).

The relationship between the palettes and the display is shown in the next page. (This is a color display.)

Graphic



All graphic figures are displayed through the palettes. This means that the number of palettes is equal to the number of colors that can be displayed simultaneously on the screen.

The color numbers are defined as shown below.

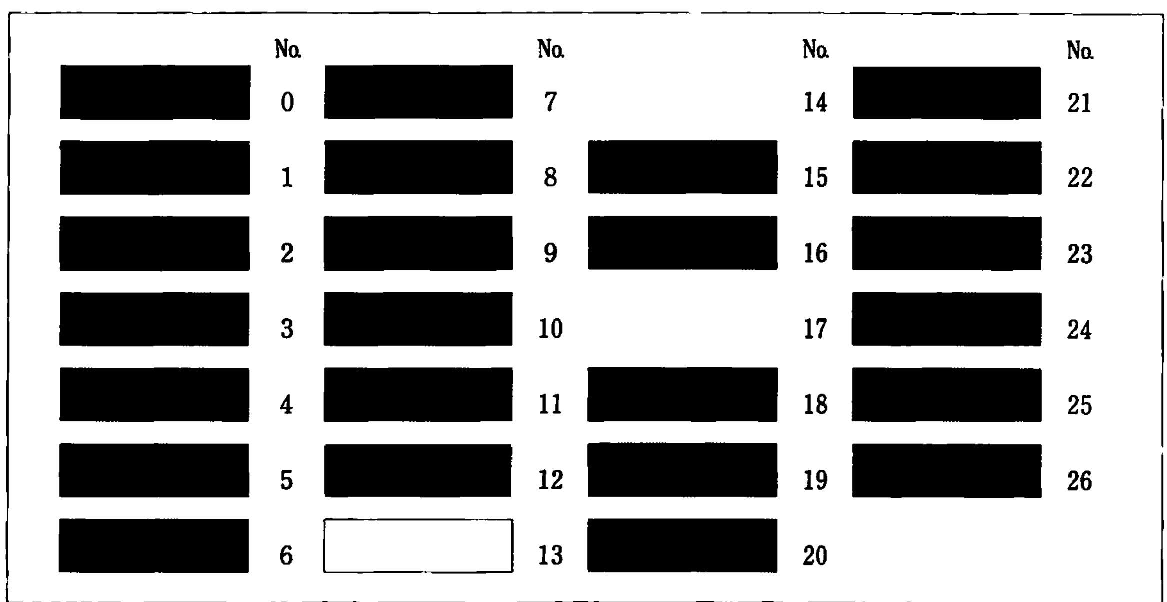
- Monochrome Display (1-VRAM specifications)

	No.
No-color	0
Colored	1

- Monochrome Display (2-VRM specifications)

	No.
No-color	0
Standard brightness	1
High brightness	2
Blinking standard brightness	27
Blinking high brightness	28

- Color Display



The initial values are automatically set to each palette when BASIC is activated.

When the color definition of a palette is modified in a program, the modification is retained during program execution. The color definition of the palette will be automatically reset to the initial values when program execution ends.

The color display contents can be printed with the A-1210 Color Printer. In this case, the color used on display will match the printout colors (approximately) only if the initial pallettes are not modified. The white portions of the screen are printed as black and the no-color (no-display) area is not printed.

The initial values of the palettes in Canon BASIC are as follows.

- Monochrome Display (1-VRAM specifications)

Palette Number	0	1
Color Number	0 (no-color)	1 (colored)

- Monochromatic Display (2-VRAM specifications)

Palette Number	0	1	2	1
Color Number	0 (No-color)	2 (High brightness)	27 (Blinking standard brightness)	1 (Standard brightness)

- Color Display

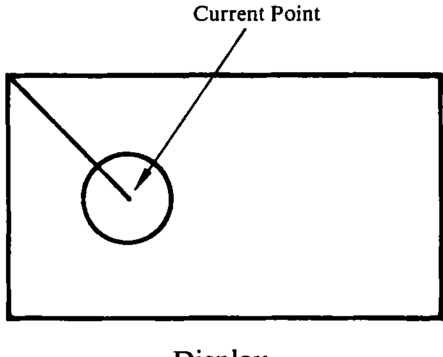
Palette Number	0	1	2	3	4	5	6	7
Color Number	0 (No-color)	1 (Blue)	9 (Red)	10 (Purple)	3 (Green)	4 (Light blue)	12 (Yellow)	13 (White)

5.1.3 Current Point

The default value of the coordinate specification in a graphic instruction is the current point.

The current point is determined by the last graphic statement executed. Its initial value is the origin (0, 0) of the absolute coordinate immediately following the start of program execution.

For example, draw a line from the coordinates (0, 0) to (150, 200) and then execute a graphic instruction for drawing a circle without specifying the coordinates of the center. The center of the circle will be (150, 200) because the current point was moved to (150, 200) when the line was drawn.



Display

5.1.4 Line Types

A line type can be specified for drawing fitures. Five line types are supported. They are numbered 0 ~ 4.

The numbers 0 ~ 4 correspond to the following line types .

Number	Line Type	
0 :	Solid line	————————
1 :	Short broken line	----------------
2 :	Long broken line	— — — — — — — — —
3 :	Single dot chain line	—·—·—·—·—·—·—
4 :	Double dot chain line	—··—··—··—··—··—

Note

Numbers exceeding 4 correspond to the line types as follows, 5 = 0, 6 = 1, 7 = 2 . . .

5.1.5 Pattern

A paint pattern in figures, i.e., rectangles, circles, fans, or an ellipses can be specified. Seven paint patterns (excluding no-paint) are supported. They are numbered 0 ~ 8. When a paint pattern in a figure is specified, the frame is drawn with a solid line regardless of the line type specification.

The numbers 0 ~ 8 correspond to the following patterns.

Number

0 :

1 :

2 :

3 :

4 :

5 :

6 :

7 :

8 :

Note

Numbers exceeding 8 correspond to the paint patterns as follows, 9=0, 10=1, 11=2. . .

5.1.6 How To Use Graphic Instructions

Like the other Canon BASIC statements, each graphic statement consists of a keyword and operands. Specify keywords and operands according to the format description for each statement.

The format descriptions of graphic statements differs slightly from those of the other Canon BASIC statements. For example, a specificable element like <Arithmetic Expression> is shown as an operand in the format description of other BASIC statements. But an element which indicates the function of a parameter, like <Coordinate> snf <Angle>, is shown as an operand of graphic statements.

Example:

LINE␣[<Coordinate 1>], <Coordinate 2>], <Coordinate 3>] ···
[␣WITH␣[<P>][, <Line Type>]]

The principal elements of the operand are:

<Coordinate>...... Specify a coordinate in the form (x, y).

<Angle> .... Specify an angle in degrees.

<P> .... Specify a palette number (0 ~ 7).

<Line Type>....... Specify a line type (0 ~ 4).

<Pattern> .... Specify a paint pattern (0 ~ 8).

Each element of operand is specified with a numeric value or an arithmetic expression. When a decimal is specified, the fractions are truncated automatically.

The sub-keyword WITH and the subsequent operands specify the graphic conditions, like the palette number, line type, and paint pattern. The following are the default values of WITH and the subsequent operands.

• <P>	Pallette number of foreground (described later)
• <Line Type>	Solid line (0)
• <Pattern>	No-paint
• <Mark>	× (4)

5.2 Graphic Declaration Instructions

5.2.1 DEFCOL Statement (Define Color)

Function

Sets colors to palettes.

Format

DEFCOL␣ [<Color$_0$>] , [<Color$_1$>] , ……, [<Color$_7$>]
①

Note 1: The comma at the end of a statement can be omitted.

Explanation

The DEFCOL statement changes the color definition of palettes. Specify up to 8 color numbers, separating them with commas (,).

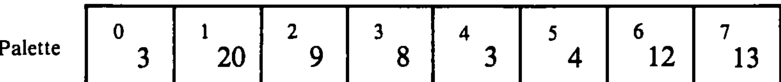
The color number specifications correspond to Palette 0, Palette 1, Palette 2, etc. This statement does not change the definition of those palettes for which a color number is not specified. The palette definition by this statement is retained only during program execution. The palette definition is reset to the initial value when program excution ends.

[Example 5.2.1-11]

Modify the definition of palettes 0, 1, 3 of the color display.

```
40 DEFCOL 3,20,,8
```

Assuming that the palette definition prior to line 40 is the initial value, the definition of the palettes 0～7 are modified as shown below by the execution of line 40.



Palette	0	1	2	3	4	5	6	7
	3	20	9	8	3	4	12	13

This definition is retained until redefinition by the DEFCOL statement is performed or until program execution ends.

5.2.2 COLOR Statement (Color)

Function

Specifies the palette for the display as the default value.

Format

```
COLOR␣<P1>[, <P2>]
```

Explanation

Output to the display is always performed through the palettes. When an output instruction without color specification (like the PRINT statement) is executed, the color of display (called the foreground) and the display field (called the background) are automatically determined by the defined palettes. Without palette specification, white characters are displayed on the no-color field on the display. This is because the system automatically specifies palette 7 as the foreground and the palette 0 as the background when the palette specification is omitted.

The COLOR statement modifies the palette definition of the foreground and the background, specifies the palette number for the foreground as <P_1>, and specifies the palette number for the background as <P_2>. The default value of <P_2> is Palette 0.

When a graphic instruction without palette specification is executed, the palette which is initially defined as the foreground is automatically specified.

The foreground and background palette definitions by the COLOR statement are retained during program execution. They are reset to the initial value when program execution ends.

For the a monochrome display, the initial value of the foreground is Palette 1 (1-VRAM) or Palette 3 (2-VRAM) and that of the background is Palette 0.

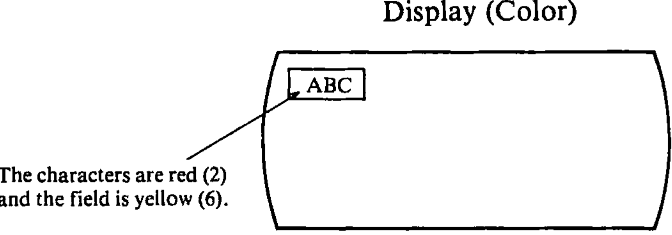
[Example 5.2.2-1]

The palette definition of the foreground and the background is modified by the COLOR statement and characters are displayed by PRINT statement.

```
10  COLOR 2,6
20  PRINT "ABC"
 :
```

If the palette color definition is not changed in the above example, the display contents are as shown below when line 20 is executed.

Display (Color)



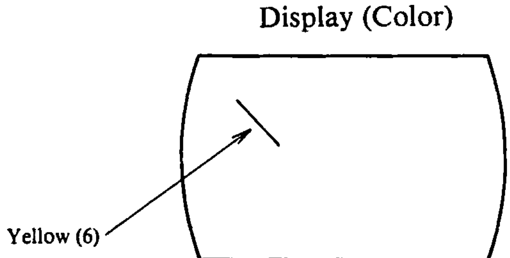
[Example 5.2.2-2]

Graphic instruction in which the palette is not specified.

```
10  COLOR 6
20  LINE (100,100),(200,200)
 :
```

In this example, the LINE statement is executed on line 20. If the palette color definition is not changed, the line is drawn on the display as follows by executing the LINE statement without palette specification on line 20. (The LINE statement is described later.)

Display (Color)



5.2.3 ORIGIN Statement (Origin)

Function

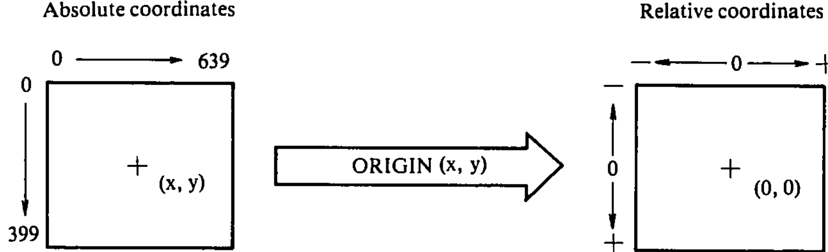
Defines the origin of relative coordinates.

Format

```
ORIGIN ␣<Coordinate>
```

Explanation

Relative coordinates specify the location of a figure that will be drawn on a display using the graphic functions. The ORIGIN statement defines the relative coordinates. The absolute coordinates which are specified as <Coordinate> will be the origin (0, 0) of the relative coordinates. When the ORIGIN statement is not executed, the absolute coordinates are the same as the relative coordinates.



After execution of ORIGIN (x_0, y_0), the relative coordinates (x, y) are the absolute coordinates ($x + y_0$, $y + y_0$).

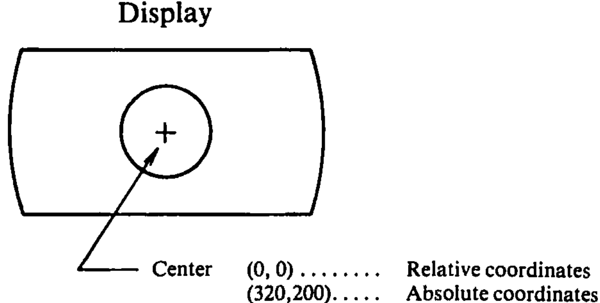
The definition of relative coordinates by the ORIGIN statement is retained during the program execution.

[Example 5.2.3-11]

Draw a circle at the center of the screen.

```
40  ORIGIN (320,200)
50  CIRCLE (0,0),100
 :
```

The ORIGIN statement on line 40 defines the absolute coordinates (320, 200) as the origin (0, 0) of the relative coordinates. The CIRCLE statement on Line 50 specifies a circle whose center is (0, 0) and radius is 100. This circle is drawn on the screen as shown below.



5.3 Graphic Drawing Instructions

5.3.1 PSET Statement (Point Set)

Function

Draws a dot.

Format

PSET␣[<Coordinates$_1$>][, <Coordinates$_2$>]··· [␣WITH␣<P>]

Explanation

The PSET statement draws a dot at the specified coordinates with the specified palette color. More than one set of coordinates can be specified.

The default value of <Coordinates$_1$> is the current point. After the execution of this statement, the current point will be the coordinates of the last dot drawn.

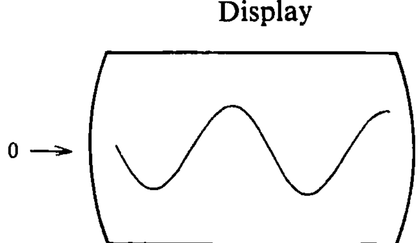
[Example 5.3.1-1]

Draw a sine curve.

```
10  PRINT %HOME
20  ORIGIN (0,200)
30  FOR I=0 TO 639
40  A=100*SIN(I)
50  PSET (I,A) WITH 5
60  NEXT I
 :
```

In the above example, a sine curve is drawn by displaying 640 dots in the direction of the x-axis. The screen is as shown below.

Display



5.3.2 LINE Statement (Line)

Function

Draws a line.

Format

```
LINE␣[<Coordinates 1>],<Coodinates 2> [, <Coordinates 3>]···
                    [␣WITH␣[<P>], [, <Line Type>]]
```

Explanation

The LINE statement draws a line of the specified line type connecting the specified coordinates. More than one set of coordinates can be specified. The default value of <Coordinates$_1$> is the current point. After the execution of the statement, the current point will be the point where the last line drawn ends.

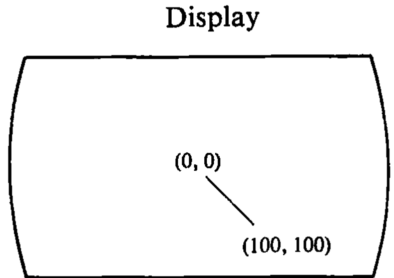
[Example 5.3.2-1]

Draw a line between the relative coordinates (0, 0) to (100, 100).

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
90  LINE (0,0),(100,100)
```

The following line is desplayed whern line 90 is executed.

Display



[Example 5.3.2-2]

Move a line by repeating line display and erasure.

```
10 PRINT %HOME
 :
50 FOR I=0 TO 639
60 LINE (0,0),(I,399)
70 LINE (0,0),(I,399) WITH 0
80 NEXT I
 :
```

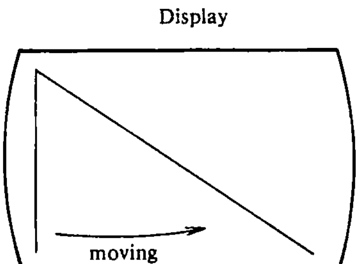
In the porgram example, the value of the x- coordinate which specifies the drawing location of the line is increased in a loop to shift the location of the line.

A line is drawn using the LINE statement of line 60. The line is erased by drawing the same line with no-color (Palette 0) using the LINE statement of line 70.

In other words, the line is drawn and then erased immediately. This operation is repeated as the location is shifted gradually. As a result, the line looks like it is moving.

As this example shows, a figure is erased by drawing the same figure with no-color (or with the field color). This example can also be used with the monochrome display because the figure is erased with no-color (Palette 0).

Display



5.3.3 RECT Statement (Rectangle)

Function

Draws a rectangle.

Format

```
RECT␣[<Coordinates 1>], <Coordinates 2>
        [␣WITH␣[<P>] [,[<Line Type>] [, <Pattern>]]]
```

Explanation

The RECT statement draws a rectangle by specifying the coordinates of its two opposite corners. The default value of one set of coordinates is the current point.

The inside of the rectangle can be painted by specifying a paint pattern. In this case, the line type will be a solid line (0) regardless of the specification. The current point is <Coordinates 1> after the execution of this statement.

[Example 5.3.3-1]

Draw a rectangle whose opposite corners are coordinates (−100, −100) and (100, 100). Specify the net pattern in the rectangle. Use Palette 6.

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
90  RECT (-100,-100),(100,100) WITH 6,,6
 :
```

The following figure is displayed by executing the previous program example.

Display

(−100, −100)

(100, 100)

5.3.4 CIRCLE Stanement (Circle)

Function

Draws a circle or an arc.

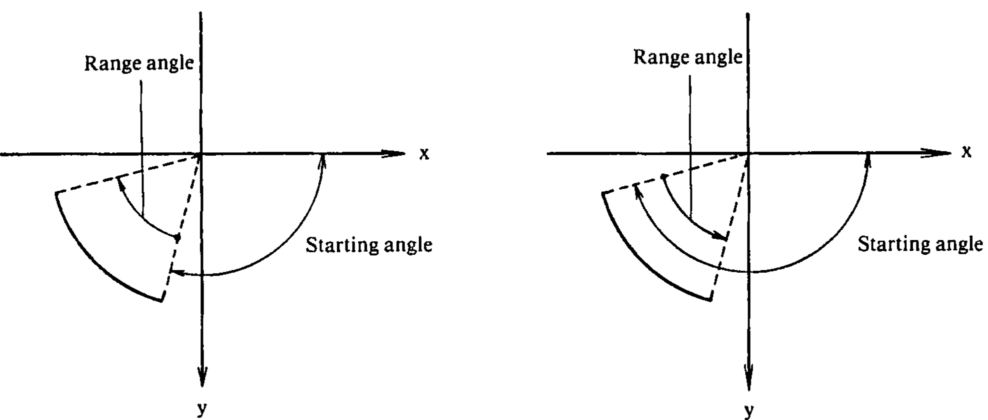
Format

```
CIRCLE␣[<Coordinates>], <Radius>[, <Angle 1>, <Angle 2>]
          [␣WITH␣[<P>][, <Line Type>][, <Pattern>]]]
```

Explanation

The CIRCLE statement draws a circle or an arc by specifying the coordinates of the center and the radius. Do not specify <Angle 1> and <Angle 2> to draw a circle.

If <Angle 1> and <Angle 2> are specified, the statement draws an arc. <Angle 1> is the starting angle indicating the starting postion of the arc. <Angle 2> is the range angle indicating the drawing range of the arc. A poxitive angle indicates the clockwise direction.



To draw an arc, specify the starting point with a starting angle and specify the degrees and the direction (clockwise direction; positive value, or counter-clockwise direction, negative value with a range angle. The starting angle must be within the range: 0 ~ 360 and the range angle must be an integer value within the range: −32768 ~ 32767. As the drawings show, some ars can be drawn by specifying starting angle = 90 and range angle = 90, or specifying starting angle = 180 and range angle = −90.

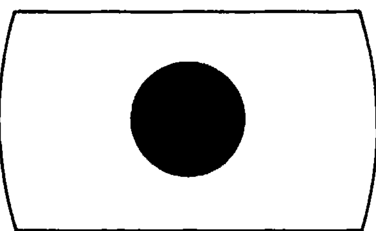
Any paint pattern specification is ignored for arc drawing. The current point is the center of the circle or arc after execution of this statement.

[Example 5.3.4-1]

Draw a circle whose center is at the coordinates (0, 0) and whose radius is 100. Paint the inside of the circle completely. Use Palette 2.

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
70  CIRCLE (0,0),100 WITH 2,,0
 :
```

The following figure will be displayed by executing the above example.

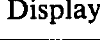


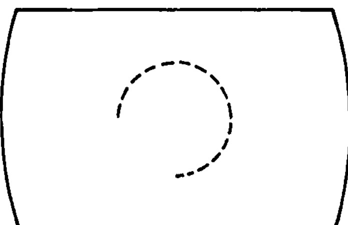
[Example 5.3.4-2]

Draw an arc using a broken line. Use Palette 2.

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
80  CIRCLE (0,0),100,90,-270 WITH 2,1
 :
```

The following figure is displayed by executing the above example.





5.3.5 FAN Statement (Fan)

Function

Draws a fan.

Format

```
FAN␣[<Coordinates>], <Radius>, <Angle 1>, <Angle 2>
              [␣WITH␣[<P>][, [<Line Type>][, <Pattern>]]]
```

Explanation

The FAN statement draws a fan by specifying operands like those for drawing an arc.

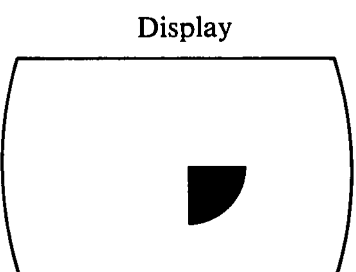
[Example 5.3.5-1]

Draw a fan and paint the inside completely. Use Palette 3.

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
70  FAN (0,0),100,0,90 WITH 3,,0
 :
```

The following figure is displayed by executing the above example.

Display



5.3.6 ELLIP Statement (Ellipse)

Function

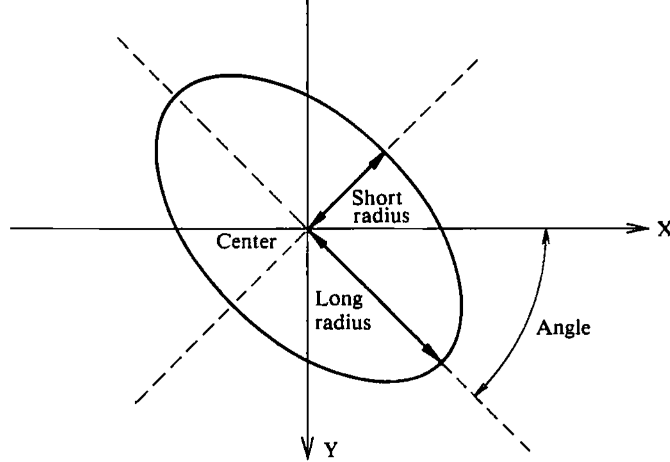
Draws an ellipse.

Format

ELLIP ␣ [<Coordinates>] , <Long Radius> , <Short Radius> [, <Angle>]
[␣ WITH ␣ [<P>][, [<Line Type>][, <Pattern>]]]

Explanation

The ELLIP statement draws an ellipse by specifying the coordinates of the center, the long radius, the short radius, and the angle. <Angle> is the angle which the long radius forms against the x – axis. It must be within the range: 0 ~ 360. The default value of <Angle> is 0.



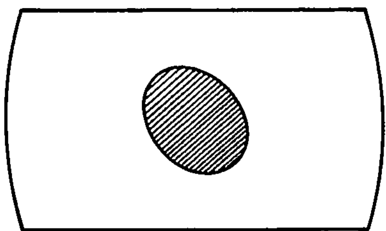
[Example 5.3.6-11]

Draw an ellipse whose center is at coordinates (0, 0), with a long radius of 200, short radius of 100, and angle of 45. Paint the inside of the ellipse using paint patterns. Specify Palette 4.

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
90  ELLIP (0,0),200,100,45 WITH 4,,2
 :
```

The following figure will be displayed by executing the above example.

Display



5.3.7 MARK Statement (Mark)

Function

Draws a mark at the specified coordinates.

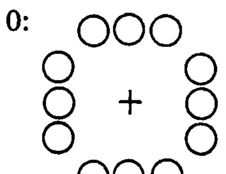
Format

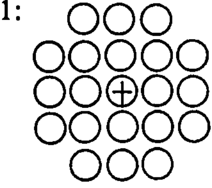
```
MARK␣[<Coordinates>][␣WITH␣[<P>][, <Mark>]]
```

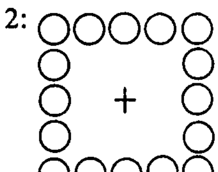
Explanation

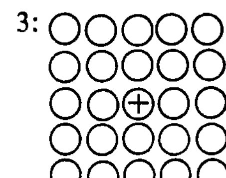
The MARK statement draws one of seven types of marks at the specified coordinates. Specify a mark number (0 ~ 6) for <Mark>. The default value of <Mark> is 4(×). The current point is <Coordinates> after the execution of this statement.

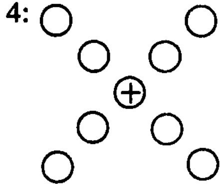
The marks and their numbers are shown below. ○ indicates a dot and + is the center of each mark.

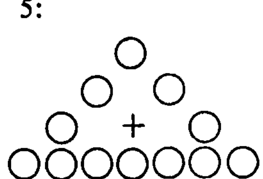
0: 

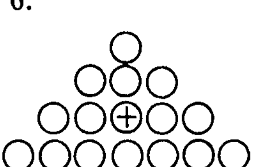
1: 

2: 

3: 

4: 

5: 

6: 

Note

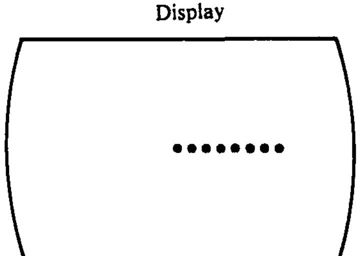
Numbers exceeding 6 coorespond to the mark type as follows,: 7 = 0, 8 = 1, 9 = 2 . . .

[Example 5.3.7-1]

Draw eight periods (.). Use Palette 6.

```
10  PRINT %HOME
20  ORIGIN (320,200)
 :
70  FOR I=0 TO 70 STEP 10
80  MARK (I,0) WITH 6,1
90  NEXT I
 :
```

The following figure is displayed by executing the above example.



5.4 Other Graphic Instructions

5.4.1 TEXT Statement (Text)

Function

Displays a character string using graphic specification.

Format

```
TEXT␣[<Coordinates>], <Character String>[␣WITH␣[<P>]
          [, [<Direction>][, <Width magnification>, <Height magnification>]]]
```

Explanation

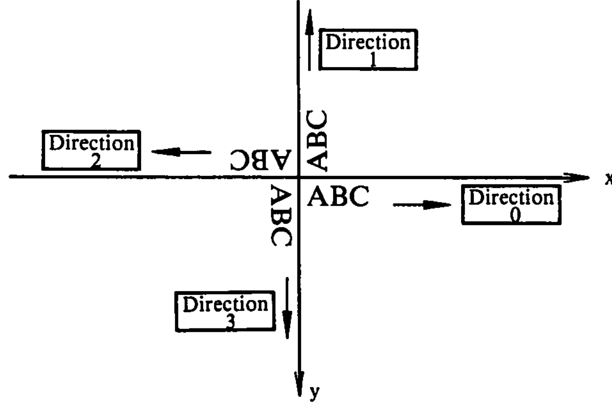
The TEXT statement draws character strings using graphic specification. Output directions and magnification of characters can be specified.

It can output the same characters that can be output by THE PRINT statement. <Coordinates> specifies the starting point of display, which is the left top of the first character of a character string. The current point is the right top of the last character after the execution of this statement.

Specify one of the following numbers, right = 0, up = 1, left = 2, down = 3 for <Direction>. The default value is 0 (right).

Specify the character magnification for <Width magnification> and <Height Magnification> with a number within the range: 1 ~ 16. The default value is 1.

These syntax rules are summarized below.



Note

Numbers exceeding 3 correspond to the directions as follows,: 4 = 0, 5 = 1, 6 = 2 . . .

[Example 5.4.1-1]

Display the character string "ABC" at a width magnification of 15 and at a height magnification of 5.

```
10  PRINT %HOME
 :
40  TEXT (0,0),"ABC" WITH,,15,5
 :
```

In the above example, the relative coordinates are the same as the absolute coordinates because the ORIGIN statement is not executed. The following character string is displayed by executing the above example.

Display

5.4.2 PAINT Statement (Paint)

Function

Paints an enclosed area.

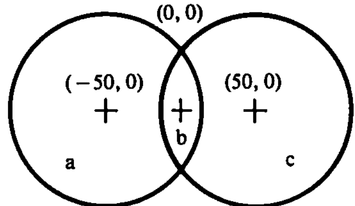
Format

```
PAINT␣[<Coordinates>][,<Boundary Palette>]
                    [␣WITH␣[<P>][,<Pattern>]]
```

Explanation

The PAINT statement paints the area containing the specified coordinates with the specified paint pattern and in the color of the specified palette. Here an area is a portion of the screen enclosed by a line(s).

For example, define the origin of relative coordinates as (320, 200) of the absolute coordinates using the ORIGIN statement. Then, draw a circle with a radius of 70 at (−50, 0) and another circle with the same radius at (50, 0). If a paint pattern is not specified in the circles, four areas, A, B, C, and D are formed on the screen as shown on the next page.

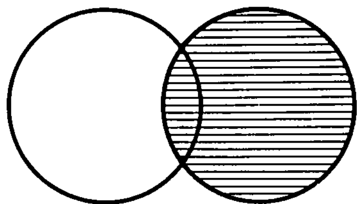


If the PAINT statement which specifies (50, 0) as the coordinates parameter is executed, area C which contains (50, 0) is painted. Areas B and A are painted by the PAINT statement which specifies (0, 0) and (−50, 0) respectively. The entire screen except the two circles (area D) is painted by the PAINT statement which specifies (0, −100).

The PAINT statement which specifies a boundary palette enables painting a figure drawn with the specified palette. For example, assume that the same circles described previously are drawn by the following two CIRCLE statements.

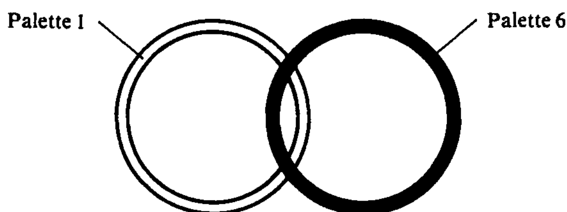
```
50 CIRCLE (-50,0),70 WITH 1
60 CIRCLE (50,0),70 WITH 6
```

The left circle is drawn with Palette 1 (blue) and the right circle is drawn with Palette 6 (red). If PAINT statement which specifies (50, 0) as <Coordinate> and Palette 6 as <Boundary Palette> is executed, area B and area C will be painted. This is because the arc between area B and area C has been drawn with Palette 1 and so is ignored by this PAINT statement. At this time, the arc connecting area B and area C is erased.



However, area A and area B cannot be painted by executing a PAINT statement which specifies (−50, 0) as <Coordinates> and Palette 1 as <Boundary Palette>.

This is because the circles drawn by the two CIRCLE statements intersect and the left circle has been partially eclipsed by the right circle. So a PAINT statement specifying Palette 1 for <Boundary Palette> treats the entire screen as one area.



When intersecting figures are drawn on the screen as explained in the previous example, the last figure will remain on the screen, so be careful when executing PAINT statements with boundary palette specification.

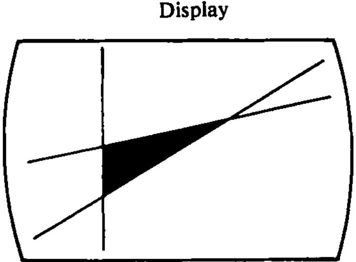
The current point is <Coordinates> after execution of the PAINT statement.

[Example 5.4.2-1]

Paint the area enclosed by three lines.

```
10  PRINT %HOME
20  ORIGIN (320,200)
30  LINE (-300,0),(300,-100)
40  LINE (-100,-200),(-100,200)
50  LINE (-300,200),(300,-150)
60  PAINT (0,0) WITH 2
```

The following figure is displayed by executing the above example.



5.4.3 GGET Statement (Graphic Get)

Function

Assigns the image data displayed on the screen to variables.

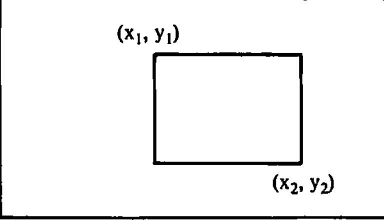
Format

```
GGET␣[<Coordinates 1>], <Coordinates 2>, <Array Variable>
```

Explanation

The GGET statement assigns the image data of a specified area on the screen to variables. The image data which are assigned to variables by this statement can be reproduced on the screen using the GPUT statement, explained later.

The image that will be written to variables as image data is specified with <Coordinates 1> and <Coordinates 2>. The area is the rectangle whose diagonal line is the line connecting these two coordinates. This is illustrated below. Either (x_1, y_1) or (x_2, y_2) may be specfied as <Coordinates 1> or <Coordinates 2>. The larger number specified as <Coodinates 1> or <Coordinates 2> is treated as (x_2, y_2). the smaller one is treated as (x_1, y_1). The default value of <Coodinates 1> is the current point.



* The frame of the rectangle is also included in the area.

The necessary number of array variables is defined according to the area of the image. The momory size (the total length of variables) is calculated using the following equation.

$$\text{Memory size (in bytes)} = 4 + N \times \lceil (y_2 - y_1 + 1) \times (x_2 - x_1 + 1)/8 \rceil$$

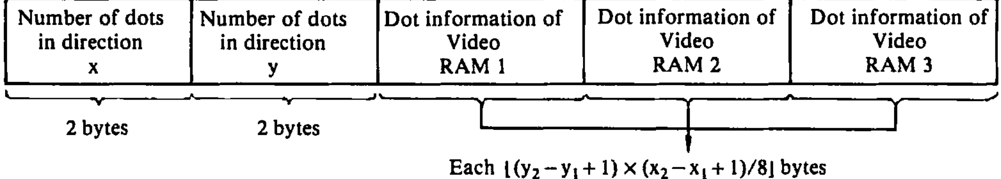
*Value of N: N=3 (Color), N=1 (Monochrome 1-VRAM), N=2 (Monochrome 2-VRAM)
* ⌊ ⌋ means to round fractions up.

Array variables may be of the real number-type, integer-type, of string-type.

The image data cannot be stored if the array variable defined doesn't have the memory size calculated above.

However, the maximum subscript value of the array is 32767. The use of real number-type variables is recommended considering the amount of image data. Therefore, the maximum subscript value of the array variable is the result obtained by dividing the calculated memory size by 8 bytes and rounding the fractions up.

The data are assigned to variables are shown below. Refer to "The CP/M-86 User's Manual" for details of the Video RAM.



Enough array variables must be defined by the DIM statement in advance to assign image data. If not, executing the GGET statement will cause an error.

The current point is <Coordinates 1> after the execution of this statement.

5.4.4 GPUT Statement (Graphic Put)

Function

Reproduces images recorded as image data on the screen.

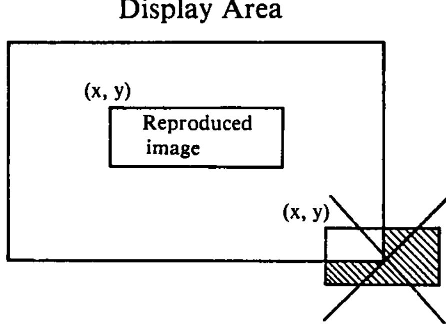
Format

```
GPUT␣[<Coordinates>], <Array Variable>
```

Explanation

The GPUT statement reproduces an image on the screen using the image data which have been assigned to variables by the GGET statement.

<Coordinates> specifies where the image will be reproduced. Specify the top left coordinate of the image area that will be displayed. If a part of the image extends beyond the screen display area, this statement is ignored.



Specify the array variables to which the image data have been assigned by the GGET Statement.

The current point is <Coordinates> After execution of this statement.

[Example 5.4.4-1]

A figure is moved using the GGET statement and GPUT statement.

```
10  DIM IMAGE(176)
20  PRINT %HOME
30  ORIGIN (320,200)
40  CIRCLE (0,0),30
50  GGET (-30,-30),(30,30),IMAGE(*)
60  GPUT (100,100),IMAGE(*)
 :
```

In the above example, the image data of a cicle drawn at the center of the screen (color display) are assigned to the array variables IMAGE using the GGET statement. Then this figure is reproduced at another position on the screen.

The image area which id recorded in variables as image data is from (−30, −30) to (30, 30), so the necessary number of the elements of the array variables is calculated by the following method.

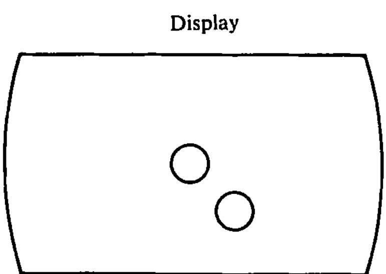
$$\text{Bytes} = 4+3\times\lfloor(30+30+1)\times(30+30+1)/8\rfloor$$
$$= 1402$$

Because real number-type array variables are used,

$$\text{Number of array elements} = \lfloor 1402 \rfloor$$
$$= 176$$

The array variables IMAGE(1) ~ IMAGE(176) are defined by the DIM statement on line 10. The following figures are displayed by executing the above example.

Display



5.4.5 CONSOLE Statement (Console)

Function

Clears the screen and specifies the screen status.

Format

```
CONSOLE ␣[{20}][, [{0}][, <Starting Line>, <Number of Lines>]]
          {25}      {1}
```

Explanation

This statement clears the contents of screen, specifies the capacity of the display lines on the screen (20 or 25), specifies smooth scroll or line scroll (0 or 1), and the range of partial scrolling (Starting Line and Number of Lines).

The capacity of display lines on the screen is the display capacity of one screen when characters are displayed using the PRINT statement, etc. The initial value is 25 lines. When this operand is not specified, the number of display lines remains the same. When the number of display lines is changed to 20, the y- coordinates of the character coordinates on the screen (coordinates specified by % CURSOR, etc.) are 0 ~ 19.

The next operand (0 or 1) specifies smooth scroll or line scroll. The unit of smooth scroll is one dot and that of line scroll is one line. The same result can be obtained using console control code [CTRL]/[2]. Specify 1 for smooth scroll and 0 for line scroll. When this operand is not specified, the scroll mode remains the same.

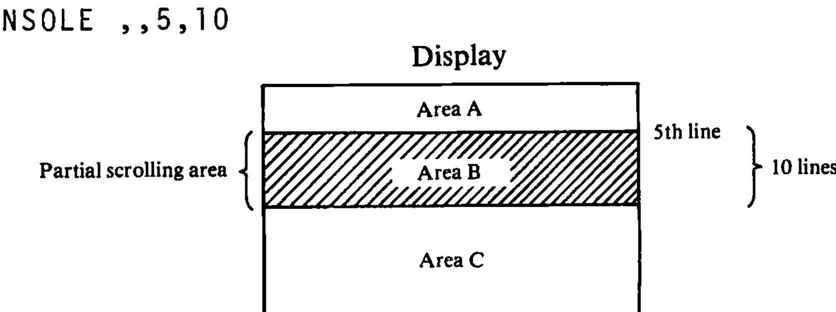
The last two operands <Starting Line> and <Number of Lines> specify partial scrolling.

The initial status of scrolling is all screen scrolling. It can be modified to partial scrolling by specifying <Starting Line> and <Number of Lines>. <Starting Line> specifies the line (counted from the top line of the screen) where partial scrolling starts. <Number of Lines> specifies the range of partial scrolling.

The display functions change as follows when partial scrolling is specified.

Assume the area of partial scrolling is specified as 10 lines starting from the 5th line by executing the following CONSOLE statement.

```
CONSOLE ,,5,10
```



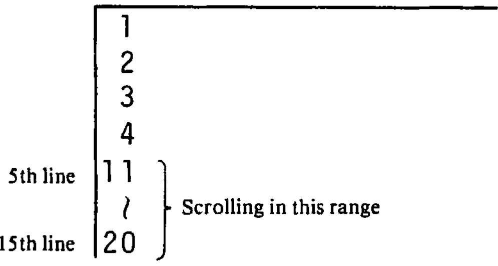
- Functions of PRINT %CURSOR(x, y), %CURX function, and %CURY
  No change.

- Functions of PRINT %HOME . . .
  If the cursor is in area A or B (1st ~ 14th line), the display contents of area A and B are erased and the cursor moves to the top left corner of the screen.
  If the cursor is in area C (15th ~ 25 (20)th line), the display contents in area C are erased and the cursor moves to the beginning of the 15th line.

- Continuous output functions using the PRINT statement . . .
  Assume that the following program is executed.

```
FOR I=1 TO 20
PRINT I
NEXT I
```

If the cursor is in area A, data is displayed normally up to the 15th line. The 16th line is not displayed and scrolling is performed in the partial scrolling area.



Scrolling is the same in the partial scrolling area when the cursor is in area B.

When the cursor is in area C, data is displayed normally through the last line of the screen.

All remaining output to the screen is displayed one after another on the last line and the final output displayed is retained as the last line on the screen.

```
15th line | 1
          | 2
          | ~
          | 9
          |10
Last line |20   <-- Output is displayed on this line one after another.
```

All specifications of the CONSOLE statement are retained until another CONSOLE statement is executed or the power is turned off.

[Example 5.4.5-1]

Specify the display capacity as 20 lines and set line scroll mode.

```
10 CONSOLE 20,0
```

Caution

All the display contents are cleared by executing the CONSOLE statement above.

5.4.6 PMODE Statement (P-Mode)

Function

Controls the pointing device and the graphic cursor.

Format

```
PMODE␣{0} [, [{0} ][, <Coordinates>]]
      {1}     {1}
```

Explanation

The PMODE statement activates the optional pointing device which controls the graphic cursor.

The graphic cursor is "+". It is moved by operating the slider on the pointing device. The position of the graphic cursor can be read by the PINPUT statement, which is explained later.

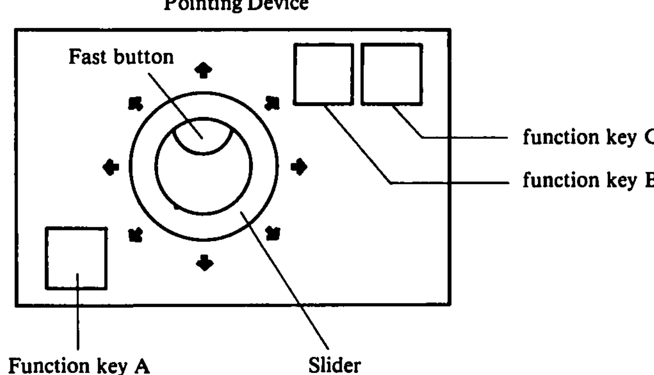
The first operand (0 or 1) of the PMODE statement specifies whether to display the graphic cursor or not. Specify 1 for graphic cursor display or 0 for no display. This operand only specifies whether the graphic cursor is displayed or not, and does not change any other functions.

The next operand (0 or 1) specifies the open/close status of the pointing device. Specify 0 to open the pointing device. In this status, the graphic cursor can be moved with the pointing device. Specify 1 to close the pointing device. In this status, the graphic cursor cannot be controlled by the pointing device.

<Coordinates> specifies the position of the graphic cursor in relative coordinates.

• Operation of the pointing device

Pointing Device



When the slider is shifted in the direction of one of the arrows, the graphic cursor moves continuously in that direction. When the fast button is pressed at the same time, the graphic cursor moves faster.

Function keys A, B, and C are defined as 18, 19, and 20 in FKEY functions. These three function keys operate regardless of the status (open/close) of the pointing device.

The graphic cursor display specification and the pointing device open/close status of this statement are retained until another PMODE statement is executed or the power is turned off. No graphic cursor display and pointing device close are automatically specified as initial values when BASIC is activated.

An example of the PMODE statement is explained with the PINPUT statement.

5.4.7 PINPUT Statement (P-Input)

Function

Reads the position of the graphic cursor.

Format

```
PINPUT ␣ <Variable 1>, <Variable 2>
```

Explanation

When the PINPUT statement is executed, the relative coordinates of the current graphic cursor position are assigned to variables. The x-coordinate is assigned to <Variable 1> and the y-coordinated is assigned to <Variable 2>.

[Example 5.4.7-1]

Move the graphic cursor with the pointing device. When function key A is depressed, draw a circle with a radius of 50 with the center at the graphic cursor position.

```
10  PRINT %HOME
20  ORIGIN (320,200)
30  PMODE 1,0,(0,0)
40  IF KEY=0 GOTO 40
50  IF FKEY(18)=0 THEN A$=INPUT$(1):GOTO 40
60  PINPUT X,Y
70  CIRCLE (X,Y),50
 :
```

When line 30 in the example is executed, the graphic cursor is displayed at the center of the screen and the pointing device is opened.

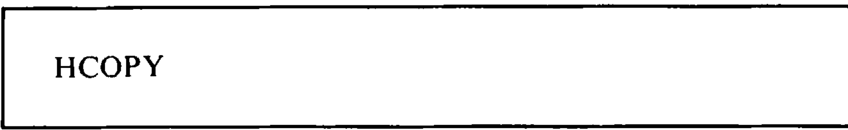
The KEY function of line 40 and the FKEY statement on line 50 confirms whether or not function key A on the pointing device is depressed. When function key A is depressed, the coordinates of the current graphic cursor position are assigned to variables X, Y by the PINPUT statement on line 60. A circle is drawn when line 70 is executed. The center of the circle is the coordinates of the current graphic cursor position.

5.4.8 HCOPY Statement (Hard Copy)

Function

Prints the display contents on the printer.

Format



```
HCOPY
```

Explanation

Executing the HCOPY statement outputs the contents of the current screen to the printer connected to I/O connector no. 1.

The printer must be the Canon Color Printer A-1210, Canon Dot Impact Printer A-1200, or another Canon-specified printer. The handler for the printer must be loaded before Canon BASIC is activated to use this statemente.

The contents of the color display can be printed with Color Printer A-1210. The colors on the screen are the same as the printout colors (approximate colos) only when the initial values of the palette color definition have not been changed. No-color on the screen is not printed and white is printed as black.)

This statement has the same function as concole control code [CTRL]/[P].

Printing using this statement can be aborted by depressing [CANEL] or [CTRL]/[C].

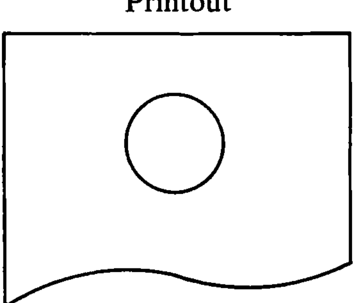
[Example 5.4.8-1]

Draw a circle on the screen and make its hard copy.

```
30 PRINT %HOME
40 CIRCLE (320,200),100
50 HCOPY
 :
```

The following figure is be printed by executing the above example.

Printout



5.4.9 POINT Function (Point)

Function

Reads the palette information of the dot at the specified coordinates.

Format

POINT(<Arithmetic Expression 1>, <Arithmetic Expression 2>)

Explanation

The POINT function is an arithmetic function and is used like other arithmetic functions.

This function checks the palette with which the dot at the specified coordinates is drawn.

When the x-coordinate of a dot is specified as <Arithmetic Expression 1> and the y-coordinate of the dot as <Arithmetic Expression 2>, this function has the palette number (0 ~ 7) of the dot specified.

For example, assuming that the dot at (100, 100) is a part of a circle drawn with Palette 2, POINT (100, 100) has a value of "2".

[Example 5.4.9-1]

Output the palette number of the dot at the specified coordinates.

```
 :
40  INPUT X,Y
50  PRINT POINT(X,Y)
 :
```

5.5 Application Examples

Various graphs can be created using the graphic functions. Examples of such programs are explained in this section.

5.5.1 Line Chart

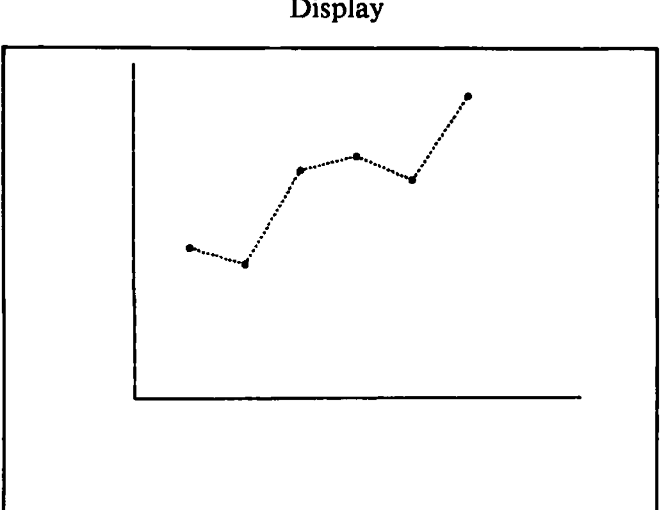
Draw a line chart using the LINE statement and the MARK statement.

```
 10 DIM DAT(6)
 20 PRINT %HOME
 30 ORIGIN (100,300)
 40 READ DAT(1),DAT(2),DAT(3),DAT(4),DAT(5),DAT(6)
 50 DATA 45,40,68,72,65,90
 60 LINE (0,-300),(0,0),(400,0)
 70 FOR I=1 TO 6
 80 MARK (I*50,-DAT(I)*3) WITH ,0
 90 NEXT I
100 PSET (50,-DAT(1)*3) WITH 0
110 FOR I=2 TO 6
120 LINE,(I*50,-DAT(I)*3) WITH ,1
130 NEXT I
140 END
```

In this example, a line chart is drawn based on six data. Points are plotted by the MARK statement in the loop of lines 70 ~ 90. These points are connected by the LINE statement in the loop of lines 110 ~ 130. The PSET statement on line 100 moves the current point to draw a line by the loop.

The line chart on the next page is drawn by executing this example.

Display



The graphic functions handle the downward direction of y-axis as positive direction. To draw an ordinary x, y graph on the screen, invert the signs by adding a minus sign (–) to the data of y-coordinates.

5.5.2 Bar Chart

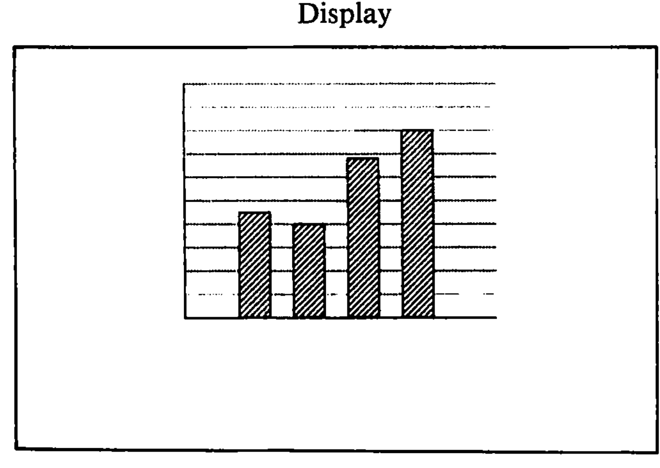
Draw a bar chart using the RECT statement.

```
 10 DIM DAT(4)
 20 PRINT %HOME
 30 ORIGIN (100,300)
 40 READ DAT(1),DAT(2),DAT(3),DAT(4)
 50 DATA 45,40,68,80
 60 LINE (0,-300),(0,0),(400,0)
 70 FOR I=1 TO 10
 80 LINE (0,-I*30),(400,-I*30) WITH ,1
 90 NEXT I
100 FOR I=1 TO 4
110 RECT (I*70,-1),(I*70+40,-DAT(I)*3) WITH ,,2
120 NEXT I
130 END
```

In this example, a bar chart is drawn from four data. The x-axis, the y-axis, and the scale are drawn by the LINE statements on lines 60 ~ 90.

The bars are drawn by the RECT statement of line 110. Parameter −1 in the RECT statement prevents the erasure of the x-axis. The following chart is drawn by executing the program example.

Display



5.5.3 Pie Chart

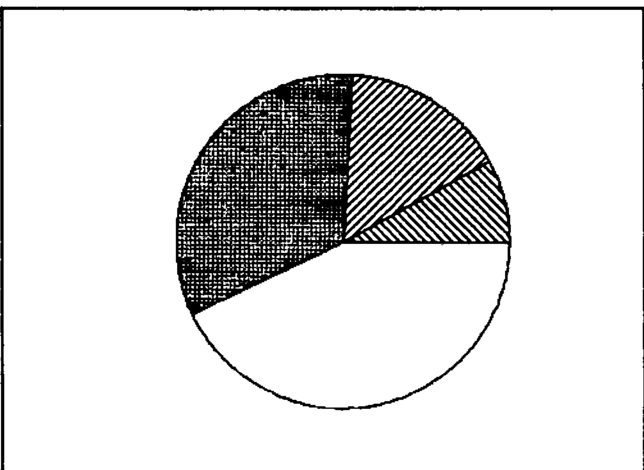
Draw a pie chart using the FAN statement.

```
 10 PRINT %HOME
 20 ORIGIN (320,200)
 30 READ A,B,C,D
 40 DATA 80,62,30,15
 50 T=360/(A+B+C+D)
 60 A=A*T:B=B*T:C=C*T:D=D*T
 70 A=FIX5(A,0):B=FIX5(B,0):C=FIX5(C,0):D=FIX(D,0)
 80 IF A+B+C+D=360 GOTO [GRAPH]
 90 IF A+B+C+D>360 GOTO [+]
100 IF A=MIN(A,B,C,D) THEN A=A+1:GOTO [GRAPH]
110 IF B=MIN(A,B,C,D) THEN B=B+1:GOTO [GRAPH]
120 IF C=MIN(A,B,C,D) THEN C=C+1:GOTO [GRAPH]
130 D=D+1:GOTO [GRAPH]
140 [+] IF A=MAX(A,B,C,D) THEN A=A-1:GOTO [GRAPH]
150 IF B=MAX(A,B,C,D) THEN B=B-1:GOTO [GRAPH]
160 IF C=MAX(A,B,C,D) THEN C=C-1:GOTO [GRAPH]
170 D=D-1
180 [GRAPH] FAN (0,0),150,0,A
190 FAN ,150,A,B WITH ,,7
200 FAN ,150,A+B,C WITH ,,2
210 FAN ,150,A+B+C,D WITH ,,3
220 END
```

In this example, a pie chart is drawn from four data.

A pie chart is drawn by combining fans. The total angle of the fans must be 360 degrees and the angle specification of each FAN statement must be an integer. Because of this, the angle of each fan is corrected on lines 90—170. When the angles of the fan total 359 degrees, 1 is added to the smallest angle. When the total is 361 degrees, 1 is subtracted from the largest angle. (The error produced by execution of line 70 is ± 1.)

The following pie chart is drawn by executing the previons example.



Graphic Instruction Formats

1. DEFCOL ␣ [<Color$_0$>] , [<Color$_1$>] ,⋯ , [<Color$_7$>]

2. COLOR ␣ <P$_1$> [, <P$_2$>]

3. ORIGIN ␣ <Coordinates>

4. PSET ␣ [<Coodinates 1>] [, <Coordinates 2>] ⋯ [␣ WITH ␣ <P>]

5. LINE ␣ [<Coordinates 1>] , <Coordinates 2> [, <Coodinates 3>] ⋯
   [␣ WITH ␣ [<P>] [, <Line Type>]]

6. RECT ␣ [<Coordinates 1>] , <Coordinates 2>
   [␣ WITH ␣ [<P>] [, [<Line Type>] [, <Pattern>]]]

7. CIRCLE ␣ [<Coordinates>] , <Radius> [, <Angle 1> , <Angle 2>]
   [␣ WITH ␣ [<P>] [, <Line Type>] [, <Pattern>]]]

8. FAN ␣ [<Coordinates>] , <Radius> , <Angle 1> , <Angle 2>
   [␣ WITH ␣ [<P>] [, [<Line Type>] [, <Pattern>]]]

9. ELLIP ␣ [<Coodinates>] , <Long Radius> , <Short Radius> [, <Angle>]
   [␣ WITH ␣ [<P>] [, [<Line Type>] [, <Pattern>]]]

10. MARK ␣ [<Coordinates>] [␣ WITH ␣ [<P>] [, <Mark>]]

11. TEXT ␣ [<Coodinates>] , <Character String>
   [␣WITH␣[<P>] [, [<Direction>] [, [<Width Magnification> , <Height Magnification>]]]

12. PAINT ␣ [<Coodinates>] [,<Boundary Palette>]
   [␣ WITH ␣ [<P>] [, <Pattern>]]

13. GGET ␣ [<Coordinate 1>] , <Coordinate 2> , <Array Variable>

14. GPUT ␣ [<Coordinates>] , <Array Variable>

15. CONSOLE ␣ [{20 | 25}] [, [{0 | 1}] [,<Starting Line> , <Number of Lines>]]

16. PMODE ␣ {0 | 1} [, [{0 | 1}] [, <Coordinates>]]

17. PINPUT ␣ <Variable 1> , <Variable 2>

18. HCOPY

19. POINT (<Arithmetic Expression 1> , <Arithmetic Expression 2>) ···Function

6. Error Messages

The error messages output by Canon BASIC have the following format:

<Error Message>! <Program Name>.<Line No.> [.<Statement No.>]

The name of the program in which the error occurs is displayed. The location of the error is indicated by line number and statement number. When <Statement No.> is 0, the 0 is omitted.
These are displayed only when errors occur during program execution.

The following error messages are displayed by Canon BASIC.

No.	Message	Meaning/Example
1	<File Name> NOT EXIST	• The specified file is not found on the disk.
2	<File Name> ILLEGAL PROGRAM	• The file specified as a BASIC program file does not have the correct BASIC program format.
3	<File Name> ALREADY OPENED	• The specified logical device number is already open.
4	<Device Name> NOT READY	• The specified dvice is not ready to use.
5	<File Name> DISK OVERFLOW	• There is not enough free area for writing on disk specified.
	<File Name> ILLEGAL RECORD NO.	• The record number specified for file assess exceeds the range 1 ~ 32767. [Ex.] PUT # 1,40000 A$

No.	Message	Meaning/Example
7	INVALID COMMAND	• The specified command is not defined.
8	NOT LOADED PFOGRAM	• The program is not loaded. [Ex.] [S][A][V][E][↵] operation when the program is not in memory.
6	FILE NOT OPENED	• The logical device number specified is not defined.
7	STOP AT	• The program was aborted by [CTRL]/[C] or [CANCEL] .
8	DECLARATION ERROR	• There is an error in declarative instructions. [Ex.] 10 DIM A (10) 20 INTEGER A Order is reversed.
9	TYPE ERROR	• There is an error in the data type. [Ex.] 3 + A$ or A$ + B
10	CONVERSION ERROR	• There is an error in type conversion. [Ex.] A = B$ B$: String variable A: Arithmetic variable

No.	Message	Meaning/Example
11	DATA ERROR	• There is an error in data read by the READ statement. [Ex.] 40 READ A, B ←Arithmetic variable 50 DATA XYZ, STU←Character data
12	ILLEGAL ARGUMENT	• There is an error in the value limitation. [Ex.] A = ASN(−2) └─ Negative value unacceptable [Ex.] OPEN # 12,"A. . ." └─ Logical device numbers must be 1 ~ 9.
13	BRANCH ERROR	• There is an error in the specification of a branch destination. [Ex.] GOTO 100 in a program that does not contain line 100.
14	MEMORY OVERFLOW	• The memory area is full.
15	ENTRY NOT FOUND	• There is an error in the specification of the entry name (keyword). [Ex.] MAT AAA(10, 10)

No.	Message	Meaning/Example
16	ADDRESS ERROR	• There is an error in the subscript of an array variable. (Ex.) 10 DIM A(5) ⋮ 40 A(10) = 100 └ Not defined
17	CONVERSION OVERFLOW	• There is an error in the conversion of a value from the real number-type to the integer-type. [Ex.] 10 INTEGER A ⋮ 50 A = 100000 └ Outside the range of integer-type constants.
18	SYNTAX ERROR	• There is a syntax error.
19	UNMATCHED NUMBER OF ARGUMENT	• The number of the argument is not matched correctly. [Ex.] PARAM A, B in program A against CALL A(A).
20	ILLEGAL FILE NAME	• There is an error in the specification of a file name.

No.	Message	Meaning/Example
21	BAD INCREMENT VALUE	• There is an error in the line interval specification of a program edit command. [Ex.] R10, 0 ↵ Line interval 0.
22	LINE NOT EXIST	• The line specified by a program edit command is not found in the program. [Ex.] C100 ↵ in the editing of a program that does not contain line 100.
23	ILLEGAL LINE NUMBER	• There is an error in the line specification of a program edit command. [Ex.] C40000 line no. must be 0 ~ 32767.
24	PROGRAM OVERSIZE	• The program size has exceeded the limitation and no more additions can be made.
25	SECURED PROGRAM	• The specified command cannot be executed because the program is secured.
26	PROGRAM NAME NOT DEFIND	• The program name is not defined.

No.	Message	Meaning/Example
27	<File Name> ALREADY EXIST	• A file with the same name as the one specified is already on the disk. [Ex.] RNAME <file 1> TO < file 2> ↑ This file is already on the disk
28	<File Name> DIRECTORY FULL	• The specified file cannot be created because the directory area of the disk is full.
29	INSUFFICIENT GGET MEMORY	• The length of array variable defined for the GGET statement is less than that required to record image data.
30	ILLEGAL DEF-FN STATEMENT	• There is an error in DEF FN statement. [Ex.] 10 DEF FNA = S(1) ↑ Undefined array variable 20 PRINT FNA * Error occurs on Line 20
31	<File Name> READ ONLY	• Data deltion or writing cannot be performed because the type of file specified is read-only. Command; SAVE, CANCEL, or RNAME Statement; PRINT, PUT, CLOSE/DEL, etc.

Besides these error messages output by BASIC, there are error messages from the operating system. Refer to "The CP/M-86 User's Manual" for these messages.

Appendix 1. Character Codes

The table below shows the character codes for display.

Code: mn

n \ m	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	NUL	DLE		0	@	P	`	p		┌		É			√	┏	0
1	SOH	DC1	!	1	A	Q	a	q	☺	└	§	Å			π	┗	1
2	STX	DC2	"	2	B	R	b	r	☻	┘	Ä	å			×	┛	2
3	ETX	DC3	#	3	C	S	c	s	♥	┐	Ö	Æ			÷	┓	3
4	EOT	DC4	$	4	D	T	d	t	♦	│	Ü	Ø			Σ	┃	4
5	ENQ	NAK	%	5	E	U	e	u	♣	─	ä	æ			↑	━	5
6	ACK	SYN	&	6	F	V	f	v	♠	┴	ö	IJ			↓	╋	6
7	BEL	ETB	'	7	G	W	g	w	●	├	ü	ij			→	┣	7
8	BS	CAN	(	8	H	X	h	x	◘	┤	ß	ò			←	┫	8
9	HT	EM	)	9	I	Y	i	y	○	┬	à	ì			µ	┳	9
A	LF	SUB	*	:	J	Z	j	z	◙	┴	°				σ	┻	10
B	VT	ESC	+	;	K	[	k	{	♂	↖	ç				Φ	┣	11
C	FF	FS	,	<	L	\	l	¦	♀	↵	é				α	─	12
D	CR	GS	-	=	M	]	m	}	¿	⇥	ù				β	┫	13
E	SO	RS	.	>	N	^	n	~	▲	⇤	è				γ	═	14
F	SI	US	/	?	O	_	o	DEL	▌	▌	ë				£	█	15
	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	

*For conversion to a decimal figure, add the two figures outside the table as follows:

$57_H \longrightarrow 7$

$\downarrow$

80

$80 + 7 = 87$

So $57_H = 87$ (decimal notation)

Appendix 2. Reserved Words

a) Keywords

LET, GOTO, GOSUB, RETURN, FOR, NEXT, IF, ON, DIM, INTEGER, DEF, INPUT, PRINT, BYE, OPTION, CALL, FREE, END, READ, DATA, RESTORE, FORMAT, REM, OPEN, CLOSE, GET, PUT, PARAM, MAT, %LOAD, %CALL, CHANGE, DEFKEY

b) Sub-keywords

%CURSOR, %HOME, %DEL, BASE, AND, OR, XOR, NOT, TO, STEP, THEN, USING, MSG, TAB, SPACE, FEED

c) Function Names

SQR, EXP, LOG, LGT, SIN, COS, TAN, ASN, ACS, ATN, FRC, RAD, DMS, ARD, ADS, SGN, ABS, INT, LEN, IDX, VER, NUM, COD, STR$, CHR$, ASC$, SIZE, MOD, FIX0, FIX5, FIX9, FIXE, MAX, MIN, TIM, PI, ERR, RND, %CURX, %CURY, KEY, EOF, INPUT$, HEX$, FKEY, COM$, TOD$

Appendix 3. Commands

a) BASIC Commands

1. EDIT[␣<Program Specification]

2. LOAD␣<Program Specification>

3. SAVE[␣<Program Specification>][,SECUR]

4. LIST[␣<Device Name>][␣<Range Specification>

5. XREF[␣<Device Name>]

6. [RUN␣][<Program Specification>][/D][;<Character String>]

7. CANCEL␣<File Specification>

8. DLIST[␣<File Specification>]

9. RNAME <File Specification 1> TO <File Specification 2>

10. NEW

11. BYE

b) OS Commands

1. COPYDISK

2. VOLCOPY

3. FORMAT

4. PIP␣<Drive Name>=<File Specification>

5. STAT

6. TOD[␣<Month>/<Day>/<Year>␣<Hour> : <Minute> : <Second>]

7. TYPE␣<File Specification>

8. BASIC [␣/<Library Name>][␣<Program Specification>
[;<Character String>]]

Appendix 4. Syntax Table

1. REM ␣ <Comment>

2. DIM ␣ <Variable> [,]···

3. OPTION BASE 0

4. INTEGER ␣ <Arithmetic Variable> [,]···

5. DEFKEY ␣ <Arithmetic Expression> , <Character Expression>

6. [LET ␣] <Variable> = { <Arithmetic Expression> | <Character Expression> }

7. INPUT ␣ [# { <1~9> | <Arithmetic Variable> } ,] [MSG (<String Expression>)] [<Variable>] [,]···

8. INPUT ␣ [# { <1~9> | <Arithmetic Variable> } ,] USING ␣ { <Line No.> | [<Label>] } <Variable> [,]···

9. PRINT ␣ [# { <1~9> | <Arithmetic Variable> } ,] [{ <String Expression> | <Arithmetic Expression> | <Sub-keyword> }] [{ , | ; }]···

   <Sub-keyword> = { SPACE (<Arithmetic Expression>) | FEED (<Arithmetic Expression>) | TAB (<Arithmetic Expression>) | % HOME | % CURSOR (<Arithmetic Expression> , <Arithmetic Expression>) }

10. PRINT ␣ [# { <1~9> | <Arithmetic Variable> } ,] USING ␣ { <Line No.> | [<Label>] } ␣ { <Arithmetic Expression> | <String Expression> } [,]···

    FORMAT ␣ <Format Specification>

11. GOTO␣ { <Line No.> | [<Label>] }

12. GOSUB␣ { <Line No.> | [<Label>] }

RETURN

13. IF␣ { <Conditional Expression> | <Arithmetic Expression> } ␣ { GOTO { <Line No.> | [<Label>] } | THEN␣<Statement> }

14. ON␣<Arithmetic Expression>␣ { GOTO | GOSUB } ␣ { <Line No.> | [<Label>] } [,]···

15. FOR␣<Arithmetic Variable> = <Arithmetic Expression 1>␣TO␣<Arithmetic Expression 2>[␣STEP␣<Arithmetic Expression 3>]

NEXT␣<Arithmetic Variable>

16. READ␣<Variable>[,]···

DATA␣ { <Constant> | <Character> } [,]···

17. RESTORE

18. END

19. BYE

20. DEF␣FN <Function Name> (<Variable>[,]···) = <Definition Expression>

21. CALL␣<Program>[(<Variable>[,]···)]

PARAM␣<Variable>[,]···

22. FREE

23. OPEN␣# { <1~9> | <Arithmetic Expression> } , { "[<Drive Name>]<File Name>" | "<Device Name>" | <String Expression> }

24. CLOSE ⌴ # { <1~9> | <Arithmetic Variable> } [, % DEL]

25. CHANGE ⌴ [MSG (<String Expression>) ,]"<Drive Name>"

26. PUT ⌴ # { <1~9> | <Arithmetic Variable> } [, <Arithmetic Expression>] ⌴ <Variable> [,]···

27. GET ⌴ # { <1~9> | <Arithmetic Variable> } [, <Arithmetic Expression>] ⌴ <Variable> [,]···

Appendix 5. Display Control Codes

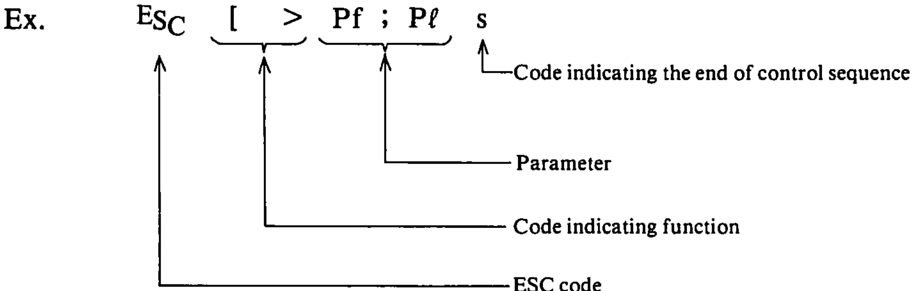
The various functions of the AS-100's display can be used by outputting specific codes to the display. These codes are called display control codes. They are supported by the CP/M-86 operating system.

This appendix explains the use of these display control codes. For further details, refer to "The CP/M-86 User's Manual."

Each display control code consists of a code string called an escape sequence or a control sequence [1]. Various functions can be performed by outputting these codes with the PRINT statement.

An escape or control sequence is a code string beginning with the ESC code ($1B_H$). The part following the ESC code specifies the function that will be used and the value (parameter) requred for its execution. Sometimes a specific code must be output at the end of a code string to indicate the end of the control code sequence.

Ex.



The parameter is specified by a number (0 ~ 9). Two or more parameters are connected by a semicolon (;).

Note 1: "The CP/M-86 User's Guide" calls a code string beginning wit ESC [a control sequence, and a code string beginning with ESC an escape sequence.

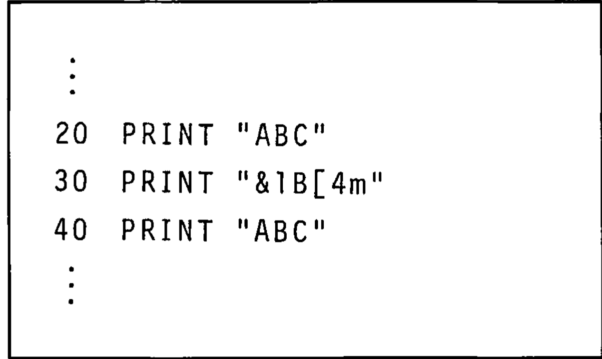
1) Specifying underlined display

Characters are underlined when they are output.

E_{SC} [4m

All characters displayed after this code is output are underlined. The specification is retained until the code to reset it is output.

[Ex.]



```
 :
20  PRINT "ABC"
30  PRINT "&1B[4m"
40  PRINT "ABC"
 :
```

Executing the program example displays the following:

Display

```
A B C
A B C
```

2) Resetting character display attributes

The underlined display specification is reset.

E_{SC} [0m

[Ex.]

```
  :
50  PRINT "&1B[4m"
60  PRINT "ABC"
70  PRINT "&1B[0m"
80  PRINT "ABC"
  :
```

Executing the program example displays the following:

Display

```
A B C
A B C
```

3) **Sound Generation**

Musical notes are generated from a speaker.

```
ESC [ > Pf;Pℓs
```

Pf: The pitch of the musical note is specified. The relationship between numbers and pitches is shown below.

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60

22 = 440 Hz (A)

Pℓ: The duration of sound is specified. The relationship between numbers and duration is shown below. The number must be within the range: 0 ~ 255.

	1	10	50	100	150	200	255
Duration (seconds)	0.016	0.16	0.8	1.6	2.4	3.2	4.08

The ratio is 1: 0.016 second.

The parameter must not exceed 30 characters including a semicolon.

[Ex.]

The C major scale is generated with one tone with a duration of 0.8 second.

```
 :
50   PRINT "&1B[>13;50;15;50;17;50;18;50s";
60   PRINT "&1B[>20;50;22;50;24;50;25;50s";
 :
```

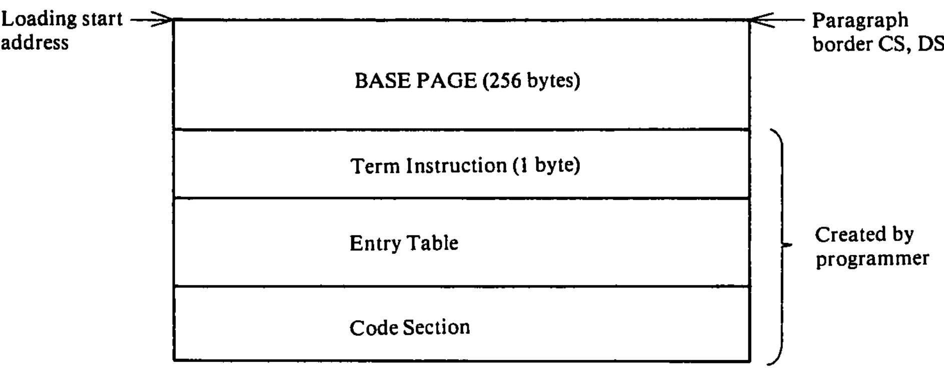
Appendix 6. Calling a Machine Language Program

It is possible to load and execute a machine language program in a Canon BASIC program using the %LOAD statement and the %CALL statement. This appendix explains how to call a machine language program.

Refer to "The CP/M-86 User's Manual" for machine language programs and more detailed explanations.

1) Structure

A machine language program must have the following structure:

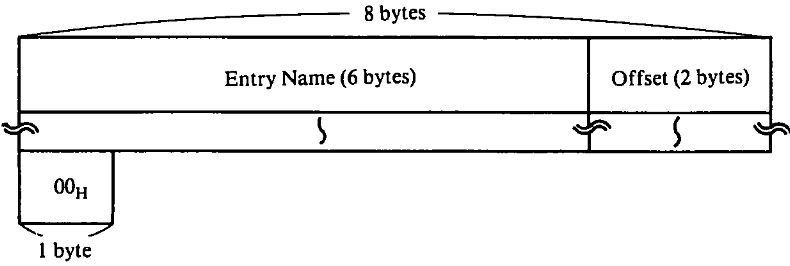


A machine language program must be created under the 8080 memory model. CS and DS are set to the loading start address.

BASE PAGE: This is created by a generator (GENCMD).

Term Instruction: This is a 1-byte FAR RET instruction to prevent problems in sole execution.

Entry Table: This has the following structure:



• Entry Name: The entry name consists of 6 capital alphabet letters and numbers starting with an capital letter. When the number of the characters is less than 6, the space is filled.

• Offset: This is an offset address from CS for the routine corresponding to the entry name.

Code Section: Processing program of each routine.

2) Execution Environment

Canon BASIC loads a machine language program specified by the %LOAD statement next to the BASIC program in memory. It searches the entry table for the entry name specified by the %CALL statement. If the entry name is found, Canon BASIC sets CS and DS to the load address and calls a routine by CALL FAR of the corresponding offset. The routine ends with FAR RET and execution returns to the BASIC program.

When a machine language program is loaded, memory clearing by the FREE statement is also valid for the machine language program. The effect of the FREE statement in this case is the same as in the BASIC program.

The formats of the %LOAD statement and the %CALL statement are shown below.

```
% LOAD ␣ <File Specification>

%CALL ␣ <Entry Name>(<Augment> [ , ] ...)
```

When the %CALL statement is executed, Canon BASIC searches the routine specified and then its offset is set to control the program, the environment is as follows:

• CS = DS indicates the loading start address of the selected machine language program.

• IP holds the offset from CS of the selected routine.

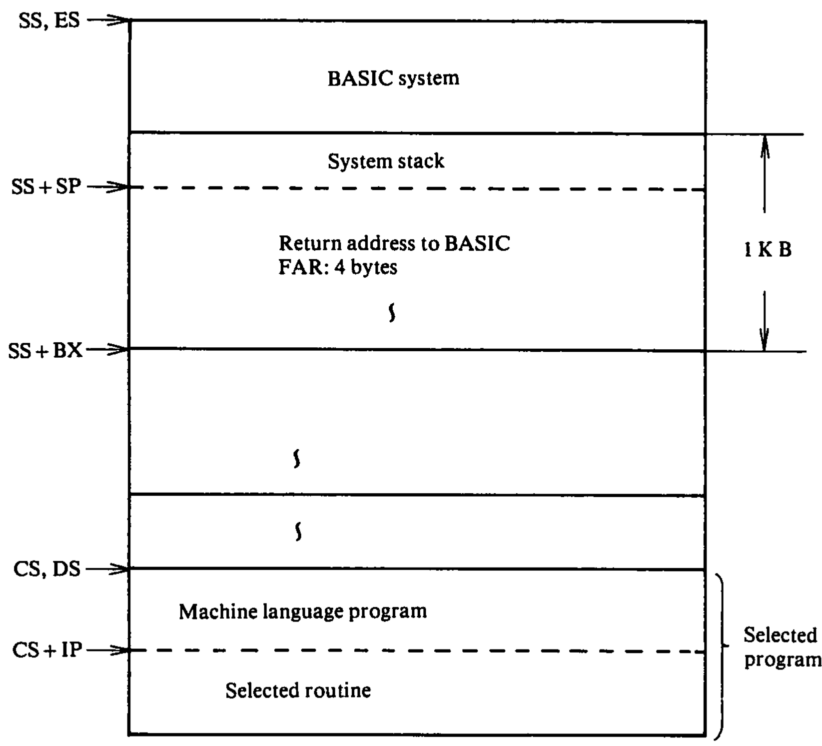
• ES = SS indicates the head address of BASIC system program.

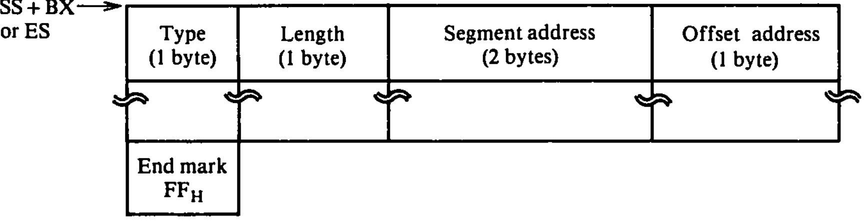
- SS = SP indicates the usable stack in BASIC. The usable stack is approximately 1 K-byte.

- SS + BX indicates the head of the argument table delivered by BASIC.

All registers can be used in machine language program execution. But when control returns to BASIC, SS and PS must be set to their original values. The contents of registers other than SS and PS do not change.

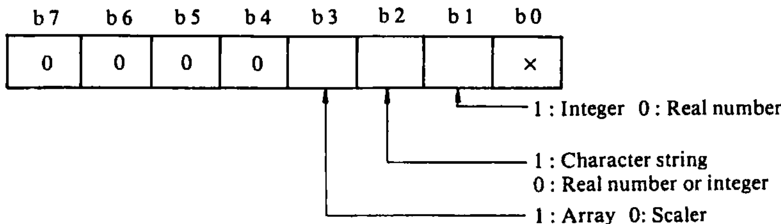
3) Registers





Type: This is the type of data delivered.

1 byte)



Note: Array indicates that all elements of the array are delivered.

Length: The length of data delivered is indicated in bytes. When the data delivered is an array, the length of one element is shown.

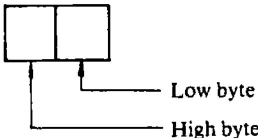
Real number → 8 bytes
Integer → 2 bytes
Character string → 1 ~ 255 bytes

Address: This is the head address where the delivered data is stored. When the data is an array, the address is the head address including the array structure information which begins the dimension of the array.

Physical address = (Sgement Address) × 16 + (Offset Address)

4) Data Structure

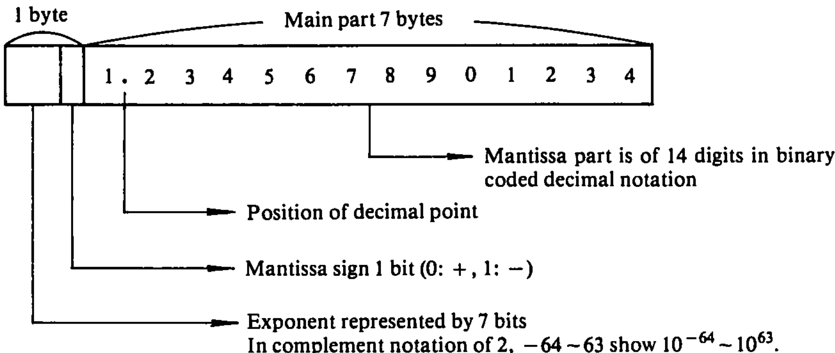
- Integer data



This is 2-byte binary data. It is stored in order of from the low-byte to the high-byte data.

Note: This is different from the ordinary notation 8088 and 8086.

- Real number data



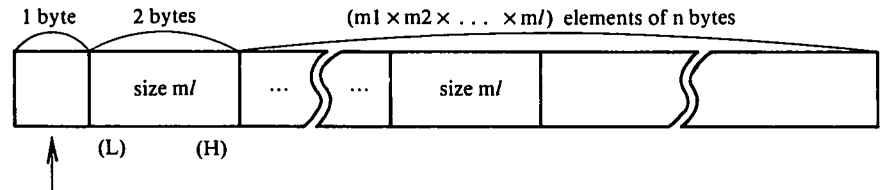
- Character data

n-byte area

* The number of characters can be specified within the range: 1 ~ 255.

* When a character string is shorter than the area length n, NUL is filled.

- Array data



* Byte length n is the data length of one element defined in the vaviable table.

* The size is set in order from the lowest byte to the highest byte.

* The total size of the array data may exceed 64KB

Note: When data exceeding the range of the data area is written or when the information part of array data is rewritten, the result is not guaranteed.