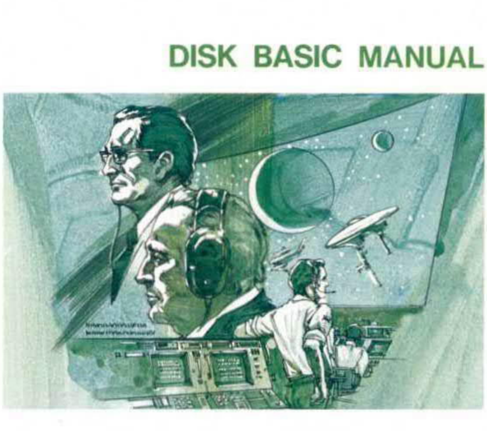
Personal Computer

MZ-700

DISK BASIC MANUAL



SHARP

DISK BASIC Manual

Introductory Note

This manual is based upon the DISK BASIC Interpreter MZ-2Z009, the system software of the MZ-700 personal computer.

(1) The DISK BASIC interpreter MZ-2Z009 includes all commands of the MZ-700 BASIC 1Z-013B. In other words, the DISK BASIC is an expansion of the BASIC 1Z-013B.

(2) For the multi-purpose MZ-700 personal computer, the system software is completely supported by a software pack (cassette tape, floppy disk, etc.) in the file form.
This system software and the contents of this manual are subject to upgrading changes for improvement, and for that reason the user is urged to particularly note the file version number. Please understand that we cannot be responsible for damage incurred during, or as a result of operation.

Introduction

We want to take this opportunity to thank you for purchasing the Sharp DISK BASIC system software.
The manual provides a general explanation of the use and programming of the DISK BASIC system software for the personal computer.
This system software is provided in the floppy disk format, and careful attention should be given to the proper use and handling of the disk drive and the disks themselves. Please refer to page 108 of this manual for information regarding the proper handling of the floppy disks.
When the floppy disk is to be used, it is recommended that the disk, a copy which is packed together with the original master disk (DISK BASIC), be used instead of the master disk. This is for protection of the master disk in the event of some unexpected trouble which might make the master disk useless. Please store the master disk in a safe place.
Before using the DISK BASIC please carefully and completely read this manual in order to assure its correct use.

■ Difference between the CASSETTE TAPE BASIC and DISK BASIC.

The following commands are extended and supplemented in the difference between the CASSETTE TAPE BASIC and DISK BASIC.

Extension	Page	Supplement	Page	Supplement	Page
RUN	36	DIR	35	XOR	65
LOAD	37	LOCK	40	NOT	65
SAVE	38	UNLOCK	41	OR	65
DELETE	39	RENAME	42	AND	65
MERGE	43	CHAIN	44		
ROPEN #	46	SWAP	45		
INPUT #	47	XOPEN #	50		
WOPEN #	48	KILL #	54		
PRINT #	49	DEFAULT	55		
INPUT # ( )	51	EOF #	56		
PRINT # ( )	52	LABEL	57		
CLOSE #	53	WAIT	58		
USR	62	SEARCH	59		
INP @	63	INIT	60		
OUT @	64				

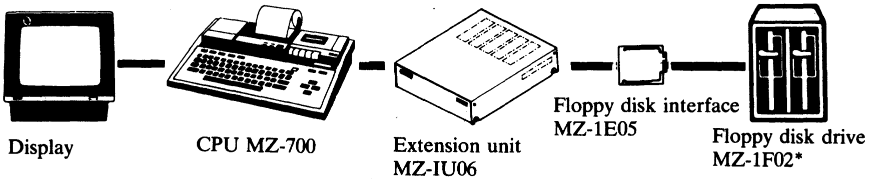
In addition to the above, there are the following differences:

CASSETTE TAPE BASIC	DISK BASIC
If the space key is pressed, the LIST display stops, and if it is not pressed, the LIST display continues.	Once the space key is pressed, the LIST display stops, and, if it is pressed for a second time, the LIST display continues.
Relative to the data file, the CASSETTE TAPE BASIC takes the format INPUT/T, etc.	Relative to the data file, the DISK BASIC takes the format INPUT #n, etc. ...
If, for any reason, the program execution is interrupted, Ready is always displayed.	If CONT is possible, Ready is displayed.

CASSETTE TAPE BASIC	DISK BASIC
CONT is possible after execution of END statement.	CONT is not possible after execution of END statement.
$\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$ has not effect.	$\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$ initialize the color and console, ~~and execute PLOT OFF~~.
Neglects space between reserved words; considers GO␣TO␣10 as GOTO␣10.	Does not neglect space between reserved words. Does not consider GO␣TO␣10 as GOTO␣10.
Compared to the CASSETTE TAPE BASIC, the user area of DISK BASIC is slightly reduced so there are some instances in which a program made in CASSETTE TAPE BASIC cannot be read in (LOAD) and executed (RUN, GOTO). Because of extension, there are some instances that programs using BASIC MONITOR with the USR function do not operate normally.The introduction of file descriptor changes the error display partially.	

■ Notes concerning the control of the floppy disk drive in MZ-700

● System composition



* Can be controlled also by MZ-700 by setting up the ROM explained on the next page.

■ Increase of floppy disk interface ROM

● Floppy disk drive control ROM

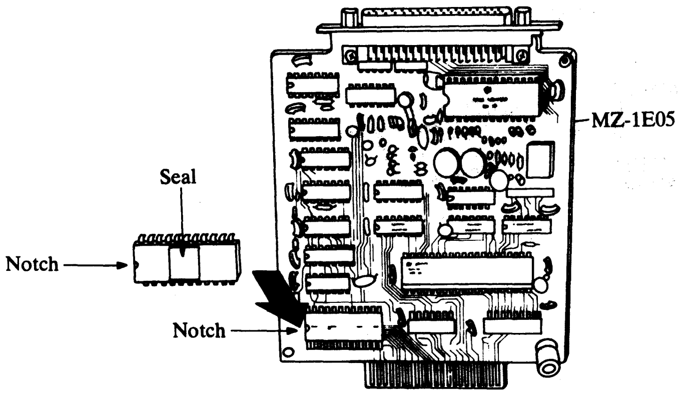
If the DISK BASIC controls the floppy disk drive (MZ-1F02), connect the ROM chip (see below) which is packed together with the DISK BASIC to the socket of the floppy disk interface (MZ-1E05: optional). If this ROM is not connected to the floppy disk interface, the floppy disk drive cannot be controlled.

Note the following points when connecting the ROM to the interface:

- Take care not to drop, scratch or otherwise damage the ROM. Do not expose it to a strong magnetic field.
- Take special care not to break or bend the pins of the ROM.
- Before pressing the ROM into the socket, be sure that its pins are correctly aligned with the socket holes.

  Be sure that the notches in the ROM and socket match each other before pressing the ROM in.

  Do not remove the seal attached to the ROM.
- Do not touch the pin of the ROM, because static electricity may destroy the ROM contents.



Contents

What the DISK BASIC is

The DISK BASIC has a strong file control function relative to the cassette base BASIC. For example, it makes the most of the features of a disk base which has high-speed read-out/write-in capabilities, and, more than simply serving for data storage, the file makes it possible to use the data area directly connected to the computer system.
Moreover, because this DISK BASIC also includes the control functions related to the RS-232C interface, it makes it possible to control various devices, by using the MZ-8BI03 serial interface (optional).
Thus, it can be said that the DISK BASIC is a new system software which permits the expansion of the software range of the personal computer.
By understanding this DISK BASIC and making full use of its many functions, the user can create a higher level system.

■ File

The computer can exchange data and programs between peripheral devices (floppy disk, cassette recorder, printer, etc.). The units of these data and programs are called files.

● File classification

There are two types of files: data files and program files.

Files
- Data files ..... These files store numbers, characters, etc. as data.
  - (BASIC sequential access data files ..... BSD
  - BASIC random access data files ..... BRD)
- Program files ..... These files store programs just as they are.
  - (BASIC text programs ..... BTX
  - Machine language programs ..... OBJ)

Computer systems can be thought of as either of two fundamental types of systems: logical internal systems composed of data-processing equipment and main memories, and external filing systems composed of processed data and program banks.

■ Data file control

There are two types of data files, depending upon the format of file access (the method of data read-out and write-in). One is called the sequential access file and the other is called the random access file.

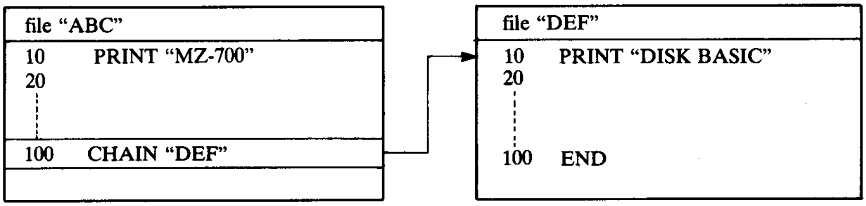
With the sequential access file, the file data access is treated as one sequential block. The file name is specified for one group of data, and such data are accessed in order from the heading at the time of registration or read-out of the file.

For the random access file on the other hand, the filed data are accessed at random. One random access file is composed of one data group designated by its own file name. Each group of data is registered in the file in a parallel arrangement, and write-in and read-out of each data group is possible by using the number (expression) assigned to that data.

If, for example, a collection of certain data can be handled as a connected group (such as, for example, data consisting of a series of decimal expressions used when producing machine language programs by BASIC POKE command and elements of tables that can be presented in order from the heading, etc.), collection of such data in a sequential access file can be useful and effective. The registration of data in a random access file, on the other hand, can be useful when it is necessary to not only consider the group of overall data but to read-out and/or write-in each element (when it is necessary, for example, to rewrite data or to search, arrange, delete, etc.).

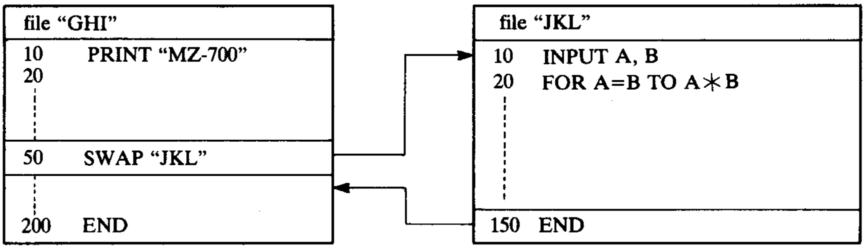
■ Program file control

The BASIC program file control commands **CHAIN** (page 44) and **SWAP** (page 45) are for read-out of another program in the memory during the execution of one program and moving the control to that program.
As shown in the figure below, CHAIN has the same function as the <goto "filename">. (For detailed information, please refer to page 26.)



CHAIN

SWAP has the same function as the <gosub "filename">. After execution of movement from the currently executing program to a separate program, it is then possible to return to the first program. (For detailed information, refer to page 27.)



SWAP

In addition it is also possible to control the various files as utility programs files and commands of the machine language program files.

Section 1
DISK BASIC OUTLINE

This section explains the features of the DISK BASIC and outlines the file controls. First of all is an explanation of the DISK BASIC starting method.

Section 2 includes a syntax explanation of the new commands and statements which the DISK BASIC has, and Section 5 includes a summary of all commands, statements, functions and operations which it has.

1.1 Starting the DISK BASIC

To make the DISK BASIC run, first perform initial loading by the IPL (Initial Program Loader). The initial loading is easily executed. With the floppy disk drive connected to the computer, switch the power supply ON and then set the disk in which the DISK BASIC is included to drive number 1 (FD1).
After making the setting, the DISK BASIC will start when the power supply of the computer is switched ON and F is input.
The figure below shows that DISK BASIC is started, and the BASIC command level condition is indicated by the flashing of the cursor.

```
  Disk Basic interpreter MZ-XXXXX VX.XX
     Copyright (C) 1984 by SHARP Corp.

  XXXXX bytes free

Ready
█
```

Note:
Please specify the default device as a cassette during write-in or read-in of a program from a cassette tape, thus starting the DISK BASIC.

```
DEFAULT "CMT:"
```

(Refer to page 55.)

■ Automatic execution of BASIC text AUTO RUN

The execution of AUTO RUN is included in the functions noted above. When the DISK BASIC is loaded and the byte size which indicates the size of the text area is displayed, the master disk is accessed once again. When the initial loading finished, the DISK BASIC automatically executes the RUN "AUTO RUN" command, i.e., the program text with the "AUTO RUN" file name is read out from the same master disk, and execution is from that heading. The program which defines the definable function

key is registered by this file name on the supported master disk.
In addition, because the NEW command is at the end of this program, "READY" is displayed after execution, and the text is erased before the cursor begins flashing. (Here, try the execution of LOAD "AUTO RUN" and check the list.)
If you want to start a certain program after the start of the DISK BASIC, the file name of that program should be saved on the master disk as "AUTO RUN."

1.2 How to Copy the Disk Basic

Start the DISK BASIC and prepare a new floppy disk for copying.

Execute the directory of DISK BASIC (DIR: refer to page 35) and "FDCOPY".

DIR "FD1:" [CR] .... Perform the directory of floppy disk drive number 1. The screen becomes as follows when this command is executed:

```
 DIR "FD1:"
 DIRECTORY OF FD1:   XXKB FREE.
  OBJ* "~~DISK BASIC MZ XXXXX~~   Deletion.
→█OBJ* "FDCOPY"
  OBJ* "TRANS"
  BTX* "AUTO RUN"
 Ready
 █
```

- Using the [↑] key, move the cursor to the position of the → sign and press the [F1] key.
  (RUN "FDCOPY")

When "FDCOPY" is executed, the screen becomes as follows.

```
            FD Format/Copy Utility VX.XX
                     [Function]
                 F ... Format disk
                 C ... Copy disk
                 I ... Boot
  ──────────────[Command area]──────────────
   *█
```

Insert the floppy disk prepared into drive number 2. When "F" is pressed, the screen becomes as follows and the drive number is asked, so specify number 2.

`*F`	

`Drive No. ?2`
`OK!`
	Input the drive number. In this instance, the drive number is 2. When the drive number is input, the floppy disk in this drive is initialized.

`*█`...............................	Indicates the end of floppy disk initialization and the waiting condition for the next command.

If there is SYSTEM software in the floppy disk inserted into the floppy disk drive, the display is as follows and confirmation is asked.

`This is Master-Disk`	
`Format [Y/N] ?N`	The meaning of display is as follows: The floppy disk inserted into the specified drive contains the SYSTEM software; is initialization and erasure OK? If it's OK, input Y; if it's not OK, input N.

*▩.................................. If N is input, changes to the condition of awaiting the next command, and, if Y is input, the floppy disk is initialized and changes to the condition of awaiting the next command.

If C is input, the screen becomes as follows:

*C .... Copies the entire floppy disk.

`Source Drive No.?1`

.... Specifies the drive of the inserted source disk (original disk) (drive number 1, in this instance).

`Destination Drive No.?2`

.... Specifies the drive of the inserted destination disk (new disk) (drive number 2, in this instance).

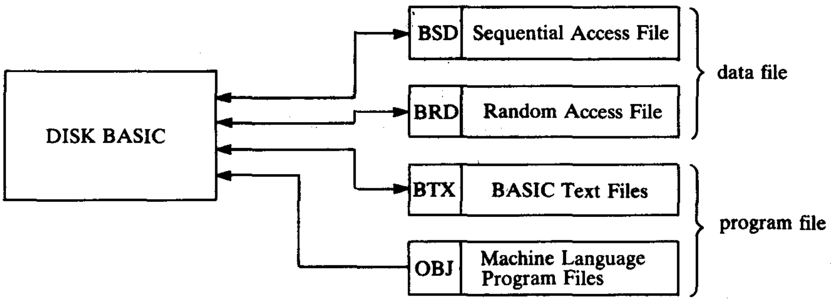
*▩ .... Indicates the end of the entire floppy disk copying and awaiting the next command.

Then the copying of the DISK BASIC is finished. The DISK BASIC is started when " ! " is input.

Note:
If the source drive and the destination drive are the same, refer to REFERENCE (page 78).

1.3 File Control

There are, as mentioned in the "File" section, 3 types of files produced by the DISK BASIC: the 2 types of data files, sequential access files (BSD) and random access files (BRD), and the BASIC text (BTX) program files. The other type of file: the machine language program files (OBJ), is a file which registers programs prepared in the MONITOR mode, etc. on the floppy disk. This can be a program by itself or it can be linked to a BASIC text as a BASIC machine language area. Thus, even though it can be used with the DISK BASIC, it is not a file to prepare and to change its contents with the DISK BASIC.



As the various file control commands are explained, first the preparation method, the use, and the features of the 2 types of data files will be explained, followed by an explanation of the use of the CHAIN and SWAP commands of the program files.

1.4 Sequential Access File Control

Sequential access files are data files in which the registration or read-out of data is in the sequential access format. The sequential access format is, as mentioned previously, a format in which access to the data is in sequential order from the heading.
The method of making a data file on a cassette file has already been explained in the BASIC manual for the MZ-1Z013B. Sequential access for the DISK BASIC is exactly the same except that the file is made on the disk rather than a cassette. Naturally, the access speed is much faster, and, because several new file control commands can be used for disk access, the breadth of useful functions for file management is also enlarged.

First, we will compare the composition of sequential access commands for DISK BASIC and for CASSETTE BASE BASIC.

File registration (data write-in)

	DISK BASIC	CASSETTE BASE BASIC
File open command	WOPEN #*n*, "*filename*"	WOPEN "*filename*"
Data write-in command	PRINT #*n*, *data*	PRINT/T *data*
File close command	CLOSE #*n*	CLOSE
Cancel command	KILL #*n*	———

File call (data read-out)

	DISK BASIC	CASSETTE BASE BASIC
File open command	ROPEN #*n*, "*filename*"	ROPEN "*filename*"
Data read-out command	INPUT #*n*, *variable*	INPUT/T *variable*
File close command	CLOSE #*n*	CLOSE
File end detection	IF EOF (#*n*) THEN	———

As you can see, comparison of the various commands shows an almost 1-to-1 relationship. Note, however, that DISK BASIC commands always include elements #n. These numbers are called logical numbers, and must always be designated for DISK BASIC file access.

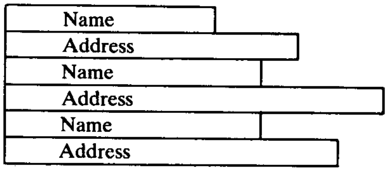
For CASSETTE BASE BASIC, file access for data write-in or read-out is limited to one file. For DISK BASIC, however, contains several files in order to make the best use of its random access feature at will, so that it is possible to simultaneously control several (**maximum 10**) files. And, if a file is opened, optionally selected logical numbers can be defined and thereafter used for designation of the pertinent file, thereby eliminating the necessity of using the file name each time.

Example:

As a simple example, let's consider the registration of a person's name and address in the sequential access file. Thus, all available addresses can be stored one after the other in the file.

Take, for example, the following file:

filename = " ADDRESS LIST "



The reason that the space used for each name and each address is of various lengths is because the data registered by sequential access are not usually of a fixed length; the length varies according to the data. For the random access files, to be explained later, all data are stored in boxes of a fixed length of 32 bytes. If, as in this example, data are to be handled as one group, and, as for the addresses above, 32 bytes is not sufficient, and the lengths are different, then the use of the sequential access file is convenient. A program can be made as described below, by INPUT command, to substitute names and addresses alternately in string variables, register each person's file individually, prepare an ADDRESS LIST of a total of 50 people, and then read out the prepared file and display on the screen the names and addresses of 10 persons at a time.

Write-in

```
100 WOPEN #3 "FD1:ADDRESS LIST" ........ Designation of drive
                                         number and file name
110 FOR I=1 TO 50
120 INPUT "name =" ; NA$
130 INPUT "address" = ; AD$
140 PRINT #3, NA$, AD$ .................. Write-in to floppy disk
150 NEXT I
160 CLOSE #3
```

Read-out

```
200 ROPEN #4, "FD1:ADDRESS LIST"
210 FOR I=1 TO 5 : FOR J=1 TO 10
220 INPUT #4, NA$, AD$
230 PRINT NA$: PRINT AD$
240 NEXT J
250 PRINT "PUSH SPACE KEY"
260 GET X$ : IF X$ = " " THEN 280
270 GO TO 260
280 NEXT I
290 PRINT "END"
300 CLOSE #4
```

■ To find the data end

What would happen if the data being read-out in order from the file surpass the number of data registered? In this case, an error does not occur. And a zero or blank is set in the read-out variable, but there is a special function EOF (#n) (page 56) which can detect the data end. When there is a data read-out by an INPUT command, EOF (#n) becomes a true condition when there is no data.
As a result, if the

```
IF EOF (#n) THEN
```

command is placed after the INPUT # command, then if EOF (#n) becomes "true", i.e. if the end of the data is found, the command after THEN will be executed.
Here use the practice problems to become sufficiently accustomed to the use of the BSD file.

Practice problems

Problem 1

Using the program example on the previous page, change the program assuming that the number of people registered is unknown, read-out the file 10 persons at a time until the end of the file is reached and display them.

Example solution

The following program might, for example, be considered.

```
300 ROPEN #5, "FD1: ADDRESS LIST"
310 FOR I =1 TO 10
320 INPUT #5, NA$, AD$
330 IF EOF (#5) THEN 400
350 NEXT I
360 PRINT "PUSH SPACE KEY"
370 GET X$: IF X$ = " " THEN 310
380 GOTO 370
400 CLOSE #5
410 PRINT "FILE END": END
```

Problem 2

Divide, and re-register, the BSD file "ADDRESS LIST" into two: a BSD file which registers names only and a BSD file which registers addresses only.

Example solution

```
500 ROPEN #6, "FD1: ADDRESS LIST"
510 WOPEN #7, "FD1: name"
520 WOPEN #8, "FD1: address"
530 INPUT #6, NA$, AD$
540 IF EOF (#6) THEN 600
550 PRINT #7, NA$
560 PRINT #8, AD$
570 GOTO 530
600 CLOSE #6, #7, #8
610 END
```

Problem 3

Register the string input by INPUT command in a BSD file. But to close the file, key input "CLOSE" and to cancel it, key input "KILL".

Example solution

```
100 WOPEN #30, "FD1: DATA"
110 INPUT "DATA = "; A$
120 IF A$="CLOSE" THEN CLOSE #30:END
130 IF A$="KILL" THEN KILL #30:END
140 PRINT #30,A$:GOTO 110
```

1.5 Random Access File Control

Random access files are data files in which the registration or read-out of data is in the random access format. The random access format is the format in which the access is done by specifying the array format.
In other words, compared to the sequential access format in which access must be from the heading of the data, the random access format can be used for access to any data in the file at random.
In order to access data in the random access file in a specified array, PRINT # (page 49) and INPUT # (page 47) are used, as described below, as expressions following logical numbers.

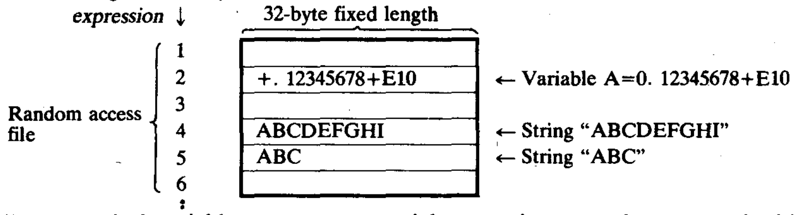
PRINT *#n (expression), data*
INPUT *#n (expression), variable*
Designation of array element

The expression is assigned by numbers or variables.
For example, the statement

INPUT #7 (21), A$

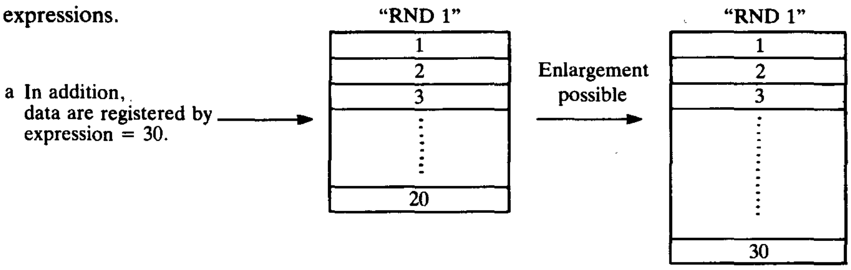
means a command to read-out, to string variable A$, the data registered as the 21st element of the data collected as the random access file opened by logical number #7.

Note that random access files in which data such as this can be accessed have the condition that all data must be registered at a fixed length, i.e., when numbers or string variables are registered in the file, they must each be set within a "box" with a limited length of 32 bytes.



For numerical variables, even exponential expressions are always stored within the 32-byte length, but, because string variables can be as long as 249 bytes, a string which exceeds 32 bytes cannot be registered in one data element of a random access file.
One other point which is different from sequential access files is that even though a file

may be once closed and in the registered condition, that same file can be made larger. The random access file "RND 1", for example, in which up to 20 expressions have been registered, can be enlarged to a file with 30 "boxes" by registering data as 30 new expressions.



Example:

Let's take an example in which, by using a random access file, a simple stock list file is to be made. Each product is assigned a fixed number from 1 to 50, and the file will have four items: product name, unit price, number in stock, total price (unit price x number in stock) and comments.

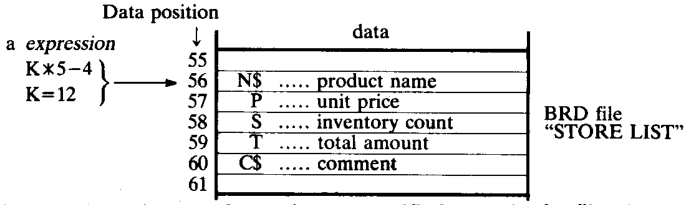
In order to register the stock data for each product, first the product number is input, and then the information to be registered for each item is input.

The program execution is ended, however, when "0" is input.

Registration of stock data

```
100 XOPEN #5, "STORELIST"
110 INPUT "product no.";K
120 IF K=0 THEN 300
130 INPUT "product name=";N$
140 INPUT "unit price=";P
150 INPUT "inventory count=";S
160 INPUT "comment=";C$
170 T=P*S
180 PRINT #5(K*5-4),N$,P,S,T,C$
190 GOTO 110
300 CLOSE #5
310 END
```

The random access file prepared in that way would be as follows. If product no. K=12, 5 data are registered in elements corresponding to array expressions 56 ~ 60.



In this way, then, data can be set in any specified array in the file. As a result, unlike the sequential access file, in which data is overlaid in order from the heading, it is also possible for there to be vacant addresses in the filed data.
And, because data can be accessed at will, it is also possible to easily rewrite data.

Use the practice problem to become accustomed to the use of the BRD file.

Problem
Let's read-out the random access file "STORE LIST" made here and read-out the inventory data for a certain product.
The program execution is ended, however, when "0" is input.

Read out of inventory data
```
500 XOPEN #7, "STORE LIST"
510 INPUT "product no.=";J:IF J=0 THEN 700
520 INPUT #7(J*5-4),N$,P,S,T,C$
530 PRINT "NO.";J:PRINT "product name ";N$
540 PRINT "unit price ";P
550 PRINT "inventory count";S
560 PRINT "total amount ";T
570 PRINT "comment ";C$
580 GOTO 510
700 CLOSE #7
710 END
```

In this way then, even for many products, the inventory data can be quickly read-out by inputting the product number.

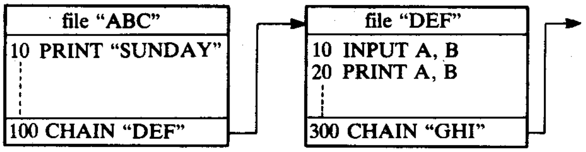
1.6 Program Chain (CHAIN)

Next, following the data file control commands, will be an explanation of program file controls. The commands explained here are CHAIN (page 44) and SWAP (page 45). When these commands are used, the program is registered on the floppy disk in job units and, while the program is being run, a separate program can be read-out and control moved to it, i.e., a program can be connected (CHAIN) to a program registered on the floppy disk, and can be read-out (SWAP) in the form of a sub routine. The CHAIN command, which connects and links the programs, will be explained first.

The form of a CHAIN command can be, for example, as follows:

```
700 CHAIN "FD1:TEXT 2"
```

This statement means that the program currently within the text area is made NEW (although the variable will be retained), the "TEXT 2" file registered on the floppy disk in drive number 1 is to be overlaid (that is, to overlay text areas and be read out), and control is moved to the heading of that text.
When this statement is executed, control will move away from the BASIC text now running, newly read out the "TEXT 2" text, and control will move to that heading.
When the program CHAIN is executed, the variable, and the function defined by DEF FN, will be transferred to the CHAIN destination program.
The CHAIN command function can be taken as <goto "filename">.



When the CHAIN command is used, a large program (even a huge program which exceeds the BASIC text area) can be divided and, as shown in the figure above, can be connected. When one program is finished, the data are left as is, and the following programs are chained one after another.
The CHAIN command can be considered indispensable if complicated and diversified data must be processed.

1.7 Program Swap (SWAP)

Program files on the floppy disk are read-in to the memory, and control is moved to this program, by SWAP command but, when this program is finished, the original program (the program which did the SWAP command) can be reset. This movement is exactly in the same way as the sub-routine in the program, and the reset to the original program becomes a return to the next command, which did the SWAP command. As a result, the SWAP command can be taken as <gosub "filename">.
In order for this operation to be made, the program which has the SWAP command is temporarily shunted to the floppy disk during SWAP execution. Then the program area is made NEW, and another program is read in. After the end of the other program, the original program is read in. The usual form of the SWAP command is:

```
SWAP "FDn : filename"
```

This is a command to SWAP the sub-program designated by the "filename" registered on the floppy disk in drive number n (n = 1 ~ 4), and the shunting of the program prior to the read-in of the other program is done on the floppy disk in the drive set by the default condition. Therefore, a floppy disk on which it is possible to temporarily write-in the program text must be set in that drive. The SWAP level must not exceed the 1 level.
In order to understand the SWAP command, let's take a simple example and follow the movement of the program file.

Program now in text area

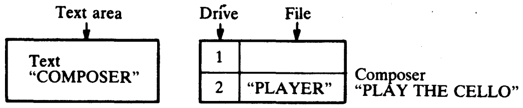
```
10  REM  COMPOSER
20  M1$="A7B6+C3 A7 A3"
30  M2$="B+C+D+E 6A3 "
40  M3$=" +F6A3+E 7 "
50  PRINT  "PLAY  THE  CELLO"
60  SWAP  "FD2:PLAYER"
70  PRINT  "VERY  GOOD"
80  END
```

Program file "PLAYER"

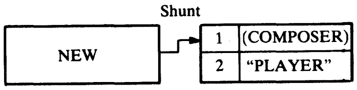
```
10  REM  CELLO  PLAYER
20  MUSIC  M1$, M2$, M3$
30  PRINT"OK?"
40  END
```

On slave disk in drive number 2

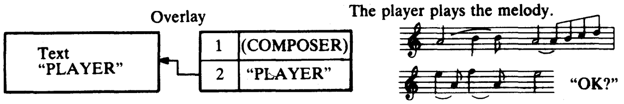
First, "COMPOSER" is within text area and is executed.



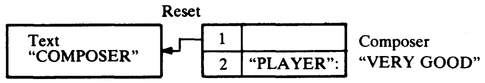
The text is first shunted to the disk in FD1 taking DIR by the SWAP command of line number 60, and the area is made NEW.



Next, BTX "PLAYER" is overlaid and executed. The melody is played.



At the end, the shunted COMPOSER is reset and says "VERY GOOD."



1.8 Reserved Words

The DISK BASIC text is composed of reserved words, also called key-words, the operand, separator and data. The DISK BASIC the reserved words, which are special words to execute certain determined functions, and commands, statements and functions are appropriate to these words.

Because the reserved words are certain words used in order to execute special commands, the programmer cannot use them as names of variables, arrays, etc. Next page shows a listing in alphabetical order of all of the DISK BASIC reserved words. (The numbers to the right of the reserved words indicate the reference page.)

When there is a (MZ) notation, refer to the Personal Computer OWNER'S MANUAL or to the DISK BASIC SUMMARY of this manual.

1.9 Table of File Input/Output Devices

Indicates the DISK BASIC descriptors

	Floppy disk	Cassette tape	RS232C
device name	FD1: ~ FD4:	CMT:	RS1: ~ RS2:
CHAIN	○	○	×
CLOSE #	○	○	○
DEFAULT	○	○	○
DELETE/RENAME	○	×	×
DIR	○	×	×
INIT	×	×	○
INPUT #	○	○	○
KILL #	○	●	●
LOAD/SAVE	○	○	×
LOCK/UNLOCK	○	×	×
MERGE	○	○	×
PRINT #	○	○	○
ROPEN #	○	○	○
RUN	○	○	×
SWAP	○	×	×
WOPEN #	○	○	○
XOPEN #	○	×	×
OPEN limit OPEN is possible for as many as 10 devices.	—	Including R/W, 1 file only	

● **Format of file descriptor**

Composed of "<Device Name> <Filename>", <Option>

1.10 Initial Settings

The default values of system variables, etc. are set as follows when the DISK BASIC is started by the IPL:

■ File descriptor

Initialization is set for the device which started the DISK BASIC.

■ Keyboard related

1) Operation mode: normal
2) Lower case character input is at the normal mode shift position.
3) The defineable function key is set as follows by BTX "AUTO RUN."

```
DEF KEY(1)="RUN   " +CHR$(13)
DEF KEY(2)="LIST"
DEF KEY(3)="AUTO"
DEF KEY(4)="RENUM"
DEF KEY(5)="DIR"
DEF KEY(6)="CHR$("
DEF KEY(7)="DEF KEY("
DEF KEY(8)="CONT"
DEF KEY(9)="SAVE   "
DEF KEY(10)="LOAD   "
```

■ CRT display related

1) Character display mode: normal (background: black)
2) Character display digit count: 40 characters/line
3) Character display scrolling area: maximum (from line 0 to line 24)

■ Internal clock

Starts at TI$="000000" initialization

■ Music functions

1) Tempo default value: (moderate tempo, moderato)
2) Default value: (quarter note, ♩)

■ Other

1) Array variables: All undefined
2) BASIC text area upper limit: number $FFFF (i.e., LIMIT MAX condition)

Section 2
DISK BASIC EXPANSION, NEW COMMANDS AND STATEMENTS

In this section each statement, function and system variable is explained. How to describe is explained in FORMAT. This symbols have the following meanings:

< > : Indicates the general description, such as, variables and data. Describes the most generic meaning.

Notes:

<variable> includes <array element>.

<variable> includes <numerical variables> and <character variable>.

<data> includes <variable> and <constant>.

[ ] : Indicates that the part enclosed by [ ] can be omitted.

[ ] ... : Indicates that the part enclosed by [ ] ... can be omitted or more than one repetition is possible.

$\left\{\begin{matrix} \text{A} \\ \text{B} \end{matrix}\right\}$: Choose one, A or B.

DIR (directory)

Displays directory contents.

Format:

DIR [/P] [<device name>]

Note: Only FDn device name.

Abbreviated form:

DI.

Explanation:

- Displays information, i.e., directory contents, relative to files registered in each device.
- When the <device name> is abbreviated, the device is considered to be designated by DEFAULT statement.
- When /P is described, the contents of the directory are output to the printer.
- The devices designated by DIR are the same ones that have been explained in the DEFAULT statement.

Examples:

DI R "F D1 : " or DIR FD1

..... Outputs on the screen the directory of the file registered in the floppy disk (drive number 1).

DI R/ P "F D1 : " or DIR/P FD1

..... Outputs to the printer the directory of the file registered in the floppy disk (drive number 1).

Reference:

DEFAULT (page 55)

RUN (run)

Executes the program.

Format:

RUN " [<device name> :] <filename> " $[\left\{\begin{matrix} <,R> \\ <,A> \end{matrix}\right\}]$

Note: Only FDn or CMT device name.

Abbreviated form:

R.

Explanation:

- When a description of the <filename> follows the RUN command, BASIC is initialized (same as NEW command), the BASIC program memorized as a file is read out, and then the program from the heading is executed, continuing to the floppy disk, etc.
- For RUN only, the program within the text area is executed.
- The specifying file is limited to the BTX or OBJ files.
- If option "A" is applied, BSD file is considered as ASCII format and is executed. And if option "R" is applied, read-in is executed putting the memory in the same condition as read-in from OBJ file IPL. This "R" option is necessary when using the OBJ program of MZ-80K series.

Example:

```
RUN  "FD1:PROG"
```

..... Reads out and executes file named "PROG" on floppy disk.

LOAD (load)

Reads out the program file.

Format:

LOAD " [<device name> :] <filename> " [<,A>]

Note: Only FDn or CMT device name

Abbreviatied form:

LO.

Explanation:

- Reads out program from external memory device.
- Specifies the file to be read out depending upon the <device name> and <filename> description.
- If the device specified by the DEFAULT statement is designated, the <device name> can be omitted.
- If option "A" is applied, BSD file is considered as ASCII format and is read in. After read-in, time is required for conversion.
- The filename must not be omitted, but from cassette tape a call out is possible even when the filename is omitted.

Examples:

```
LOAD "FD1: MZ-700"
```

..... The file named "MZ-700" is read out from the floppy disk (drive number 1).

```
LOAD "CMT: MZ-700"
```

..... The file named "MZ-700" is read out from the cassette tape.

SAVE (save)

Registers the program as a file.

Format:

SAVE " [<device name> :] <filename> " [<,A>]

Note: Only FDn or CMT device name.

Abbreviated form:

SA.

Explanation:

- Registers the program as a file on the floppy disk, etc.
- If the device designated by the DEFAULT statement is specified, the <device name> can be omitted.
- If option "A" is applied, the SAVE is as ASCII format.
  The file is saved as a BSD file, so take care not to assign a filename which would be confused with the data file made by WOPEN # and PRINT # statements.
- The file type created by the SAVE command is a BTX or BSD file.
- The <file name> cannot be omitted.

Example:

```
SAVE  "FD1:PROG"
```

..... Puts the filename "PROG" in the program on the floppy disk (drive number 1) and registers it. The type of registered file becomes BTX.

DELETE (delete)

Deletes designated file.

Format:

DELETE " [<device name> :] <filename> "

Note: Only FDn device name.

Abbreviated form:

D.

Explanation:

- When the device name is omitted, processing proceeds relative to device designated by DEFAULT statement.

Example:

```
DELETE "FD1:SAMPLE"
```

..... Deletes “SAMPLE” file of floppy disk (drive number 1).

LOCK (lock)

Perform the protection (lock) of the file.

Format:

LOCK " [<device name> :] <filename> "

Note: Only FDn device name.

Abbreviated form:

LOC.

Explanation:

- If the file is locked, this file is fixed on the floppy disk and it will not accept any change commands. For instance, DELETE and RENAME commands and data write-in are prohibited. Put a lock on any file which you don't want to destroy or change.
- The " * " mark is displayed before the file specs in the directory display on locked files.

  For example:

  `LOCK "SAMPLE"`

  If the command above is executed, the file is locked and the directory display is as follows:

  ~~LOCK~~ * "SAMPLE"

  BTX └─ sign to indicate file locking.
- Locking can be cancelled by the UNLOCK statement.

Example:

`LOCK "FD1:SAMPLE"`

..... Locks the file named SAMPLE on file set in drive number 1.

UNLOCK (unlock)

Unlocks the specified locked file.

Format:

UNLOCK " [<device name> :] <filename> "

Note: Only FDn device name.

Explanation

- Unlocks the specified locked file.

Example:

```
UNLOCK  "SAMPLE"
```

..... releases the lock of file named SAMPLE

RENAME (rename)

Changes the filename.

Format:

```
RENAME "[<device name>:]" " <old filename> "," <new filename> "
```

Note: Only FDn device name.

Abbreviated form:

RENA.

Explanation:

- The filename change specifies the current filename and the new filename, in that order.
- The new filename becomes an error if a file of the same name exists in that device.

Example:

```
RENAME  "FD1:OLDPROG", "NEWPROG"
```

..... The filename "OLDPROG" in the floppy disk file is changed to "NEWPROG".

MERGE (merge)

A program in the file is added to a program in the memory.

Format:

MERGE [" [<device name> :] < filename > "] [<,A>]

Note: Only FDn or CMT device name.

Abbreviated form:

ME.

Explanation:

- Reads in the program in the file designated by the <device name>, adding it after the program currently in the memory.
- If the <device name> is omitted, the file of the device designated by the DEFAULT statement will be read in.
- If line number of the program within the computer is the same as the line number of the program read in from the file, the program from the file has priority, and the former program will be deleted.
- If option "A" is applied, the BSD file is considered as ASCII format and is merged.

Examples:

Program in the memory	**"PROG" program on floppy disk**
10 B=2	10 A=1
30 PRINT B	20 PRINT A
50 END	40 END

..... when these are merged by MERGE "FD1: PROG", the result is as follows.

```
10  A=1
20  PRINT  A
30  PRINT  B
40  END
50  END
```

CHAIN (chain)

Movement of execution from active program to program in file

Format:

CHAIN " [<device name> :] <filename> "

Note: Only FDn or CMT device name.

Abbreviated form:

CH.

Explanation:

- The execution of the program is moved from the currently active program to a different program in the file.
- The CHAIN statement is also considered to be a file opening.
- Although the CHAIN statement has a function similar to the execution of the RUN command in a program, the variables, arrays, etc. of the original program are transferred to the new program without change when there is a chain.

Example:

```
10  A=1
20  B=2
30  CHAIN  "FD1:  PROG"
40  END
```

..... The program of the file named "PROG" on the floppy disk in the floppy disk drive (drive number 1) is executed.
In this instance, the value of variables A and B don't change; A=1, B=2.

SWAP (swap)

Calls out a program in the file during execution of the program.

Format:

SWAP " [<device name> :] <filename> "

Note: Only FDn device name.

Abbreviated form:

SW.

Explanation:

- Subroutine calls separate program in file from program being executed. Program in program area is temporarily evacuated to vacant area in floppy disk, and called program is read into text area. The original variable in the text area at this time is transferred to the called program. (See page 27.)
- An error occurs if a write-protect seal is put on the default drive floppy disk.

Example:

```
10  A=1 : B=2
20  PRINT  "A="; A, "B="; B
30  SWAP  "FD2 : PROG"
40  PRINT  "A="; A, "B="; B
50  END

[PROG]
10  A=A*10 : B=B*10
20  END
```

(SWAP FD2 : PROG)

..... When this program is executed, A = 1 and B = 1 are displayed on CRT screen when line 20 is executed. At line 30, the SWAP command is executed and the program file "PROG" in floppy disk drive 2 is executed. At end, automatically returns to original program, A = 10 and B = 20 are displayed when line 40 is executed, and program execution finishes.

ROPEN # (read open)
Opens the BSD file for read-out.

Format:
ROPEN # <logical number>, "[<device name> :] < filename> "

Note: Only FDn, CMT, or RSn device name.

Abbreviated form:
RO. #

Explanation:
- Opens the file so that filed data can be read out.
- The ROPEN # statement has an order for data read-out; it specifies the file to be read out, specifying by <device name> and <filename>.
- When the <device name> is omitted, the device designated by the DEFAULT statement is specified.
- When "RS $\begin{Bmatrix} 1 \\ 2 \end{Bmatrix}$:" is specified, input is specified to RS-232C.

Examples:
```
10  ROPEN  #1,  "FD1:DATA"
```
..... The floppy disk's (drive number 1) BSD file "DATA" is read out.

```
10  ROPEN  #1,  "RS1:"
```
..... The input by INPUT command is set to RS-232C.

```
10  ROPEN  #2,  "DATA"
20  FOR I  =1  TO  99
30  INPUT  #2, A
40  PRINT  A
50  NEXT  I
60  CLOSE  #2
70  END
```

..... The file made by WOPEN # command is read out and data are displayed.

References:

INPUT #	(page 47)
WOPEN #	(page 48)
CLOSE #	(page 53)
EOF (#)	(page 56)

INPUT # (Input)

Reads out data from BSD file.

Format:

INPUT # <logical number>,<variable>,<variable>, ...

Abbreviated form:

I. #

Explanation:

- Reads out data in sequence from heading of file opened for read-out by ROPEN # statement and is set to <variable>.
- <variable> may be array element.
- The file which reads out the data becomes the file set to <logical number> by ROPEN # statement.
- In the same way as for READ ~ DATA statements, error is generated if data and <variable> data type do not coincide.
- The end of the file data can be determined by the EOF function.
  However, for FD device only.

Example:

```
10  ROPEN  #2, "DATA"
20  INPUT  #2,A,B,C
30  PRINT  A,B,C
40  CLOSE  #2
50  END
```

..... Reads out numerical data from BSD file opened for read-out by logical number 2, and substitutes to numerical variables A, B and C.

References:

ROPEN # (page 46)
CLOSE # (page 53)
EOF (#) (page 56)

WOPEN # (write open)

Opens the files for write-in.

Format:

WOPEN # <logical number>, " [<device name> :] <filename> "

Note: Only FDn, CMT, or RSn device name.

Abbreviated form:

WO. #

Explanation:

- This is a statement to prepare for file write-in; it specifies the file's logical number and name.
- If the <device name> is omitted, processing occurs relative to the device specified by the DEFAULT statement.
- When RS is specified for the device name, the filename can be omitted.

Examples:

```
10  WOPEN  #1, "FD1:DATA"
```

..... Opens the "DATA" file defined to logical number 1 for write-in.

```
10  WOPEN  #1, "RS1:"
```

..... Output by the PRINT # command is set in RS-232C.

```
10  WOPEN  #2, "DATA"
20  FOR I =1  TO 99
30  PRINT  #2, 1
40  NEXT I
50  CLOSE  #2
60  END
```

..... 1 ~ 99 count is written into the file.

References:

PRINT # (page 49)
ROPEN # (page 46)
CLOSE # (page 53)
INPUT # (page 47)

PRINT # (print)

Writes data in BSD files.

Format:

PRINT # <logical number>, <data> [, <data>] ...

Abbreviated form:

R. #

Explanation:

- Writes data in order into files opened by the WOPEN # statement for write-in.
- Files into which data are to be written are to be correctly specified by the logical number when opened.
- (Data) are numerical value data or character data.

Example:

```
10 WOPEN #1,"DATA"
20 PRINT #1,1,2,3
30 CLOS #1
40 END
```

..... Numerical data 1, 2 and 3 are written in the file that has been opened to write in logical number 1.

References:

WOPEN # (page 48)
CLOSE # (page 53)

XOPEN # (cross open)

Opens BRD file for read-out and write-in.

Format:

XOPEN # <logical number>, " [<device name> :] <filename> "

Note: Only FDn device name.

Abbreviated form:

X. #

Explanation:

Opens the random access data file (BRD), and opens (cross opens) the read-out/write-in of the random access data. Newly registers BRD files, prepares for data read-out from, and new data write-in to, the BRD file, and defines the logical number for file access.

Examples:

```
10  XOPEN  #1, "FD1: DATA"
20  INPUT  "product  no. =";K
30  IF  K=0  THEN  110
40  INPUT  "product  name =";NS
50  INPUT  "unit  price=";P
60  INPUT  "stock  amount ="S
70  INPUT  "comment =";CS
80  T  =  P*S
90  PRINT  #1  (K*5-4), N$, P, S, T, C$
100  GOTO  20
110  CLOSE  #1
120  END
```

..... Cross opens (read-out, write-in) BRD files. The 5 items of input (product name, unit price, amount in stock, total amount (T = total amount (P ∗ S)), and comment are registered in the BRD file "DATA".

INPUT # (   )(input)

Reads out data from BRD file.

Format:

INPUT # <logical number>, (<data element number>), <variable> [,<variable>,...]

Abbreviated form:

I. # (   )

Explanation:

- The data of the position designated by the data element number are read from the file cross-opened by the XOPEN # statement into the designated variable. It makes no difference if the variable is an array variable.
- The file which reads out the data becomes the file set to <logical number> by the XOPEN # statement.
- In the same way as for the READ ~ DATA statements, an error will be generated if the data and the <variable> data type do not coincide.
- The end of the file data can be detemined by EOF function.
- The data element position is from 1 ~ n.

Example:

```
10  XOPEN  #2, "FD1:DATA"
20  Z=3
30  INPUT  #2(Z), 10
40  IF  EOF  (#2)  THEN  80
50  PRINT  S
60  Z=Z+5
70  GOTO  30
80  CLOSE  #2
```

Data element position →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	---	---
Data contents →			①					②					③		---	---

..... When this program is executed, the data at positions ①, ② and ③ data positions indicated by Z in the figure are read in and displayed in order. When the data end, the EOF # function becomes truth at line 40, and jumps to line 80.

Reference:

PRINT # (   ) (page 52)

PRINT # (   ) (print)

Writes in data to BRD files.

Format:

PRINT # <logical number>, (<data element number>), <data> [, <data>, ...]

Abbreviated form:

P. # (   )

Explanation:

Writes in contents specified by variable or numerical value to data position specified by data element number from file cross opened by XOPEN # statement. Variable may be array variable.

The file which writes in the data becomes the file specified to (logical number) by the XOPEN # statement.

The data are numerical data or character data.

The possible write-in data length is 32 bytes and the data element position is from 1 ~ n.

Example:

```
10  XOPEN  #4,  "FD1:DATA"
20  FOR  Z=3  TO  13  STEP  5
30  PRINT  #4  (Z), 0
40  NEXT  Z
50  CLOSE  #4
```

Data element position →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	---	---
Data contents →			①					②					③		---	---

..... When this program is executed, data "0" are written in at the ①, ② and ③ data position indicated by Z, as shown in the figure.

Reference:

INPUT # (   ) (page 51)

CLOSE # (close)

Closes the file.

Format:
CLOSE [# <logical number>]

Abbreviated form:
CLO. #

Explanation:
- Closes the data file opened by the logical number; returns this number to the non-defined condition.
- CLOSE relative to XOPEN #
  Closes the BRD file opened for read-out/write-in and returns all logical numbers used to the non-defined condition.
- CLOSE relative to WOPEN #
  Registers formally the BSD file opened for write-in; and returns the logical number used to the non-defined condition.
- CLOSE relative to ROPEN #
  Closes the BSD file opened for data read-out; and returns the logical number used to the non-defined condition.
- When logical number not specified, closes all currently open files, and returns all logical numbers to non-defined condition.
- When do not specify logical number, closes all currently open files; logical numbers are returned to non-defined condition.

Examples:
```
10  CLOSE#1
```
..... Closes logical number #1 file.

```
20  CLOSE
```
..... Closes all files.

References:
XOPEN # (page 50)
WOPEN # (page 48)
ROPEN # (page 46)

KILL # (kill)

Stops file registration.

Format:

KILL [# <logical number>]

Abbreviated form:

KI. #

Explanation:

- Stops registration of the file opened to <logical number>. In other words, the file making is prepared by the XOPEN # or WOPEN # statement, or stops, during registration, the formal registration of the file thereafter executing the PRINT # ( ) or PRINT #.
- If the <logical number> is not designated, the formal registration of the file currently being made is stopped and, in addition, all files already opened are closed and all <logical numbers> are returned to the non-defined condition.
- Because the KILL command can be used as a direct execution command, this command can be executed directly prior to exchange of the cassette tape, floppy disk. In order to thereby protect the contents of all files by closing them.

Examples:

```
200 KILL #3
```

..... Kills the file opened by the logical number 3.

```
300 KILL
```

..... Kills all files opened.

Reference:

XOPEN # (page 50)

WOPEN # (page 48)

DEFAULT (default)

Sets device names.

Format:

DEFAULT "<device name>: "

Note: Only FDn, CMT, or RSn device name.

Abbreviated form:

DEF.

Explanation:

- Defines available device name if <device name> is omitted by command statement.
- The device name reading DISK BASIC is defined in FD1.
- Device names are as follows.

FDn	:	floppy disk ($n = 1 \sim 4$)
CMT	:	cassette tape
$RS\begin{Bmatrix}1\\2\end{Bmatrix}$	:	RS-232C interface ($n = 1 \sim 2$)

Examples:

```
DEFAULT  "CMT: "
```

..... Considered to be cassette tape if <device name> is omitted in each command.

```
DEFAULT  "FD2:"
```

..... Considered to be floppy disk (drive 2) if (device name) is omitted in each command.

EOF (#) (end of file)

Function to locate end of data file data.

Format:

EOF (# <logical number>)

Abbreviated form:

EO.(#

Explanation:

- The function used to determine the end of the data for file read-out.
- When read-out continues after the end of data in the data file, there is no error generation, and 0 or null (“ ”) is provided as the data value.
- Error is generated, however, when data are read out from CMT.
- This function is used in combination with the IF statement and placed after each INPUT statement.

Example

```
10  ROPEN #3,"DATA"
20  INPUT#3,A
30  IF EOF(#3)THEN END
40  PRINT A
50  GOTO 20
```

..... With this program data are read-out in sequence from the “DATA” file and are displayed on the screen. And, if the read-out data are out of file, this process should be end.

LABEL (label)

Sets the label.

Format:

LABEL " <label name> "

<label name> : character line to 249 characters

Abbreviated form:

LA.

Explanation:

- It is possible to set a label which expresses the branch destination of the GOTO statement, GOSUB, etc. In this way, the program can be made easier to see and easier to understand.

Example:

```
10 PRINT "SAMPLE"
20 GOSUB "ABC"
30 PRINT "END"
40 END
100 LABEL "ABC"
110 PRINT "LABEL SAMPLE"
120 RETURN
```

..... The line number 20 GOSUB statement branch destination is set to the "ABC" label on line number 100. As a result, the subroutine after the line number 100 is called out by the GOSUB statement.

WAIT (wait)
Interrupts the program execution for a definite time.

Format:

WAIT <numerical data>

Abbreviated form:

W.

Explanation:

- Interrupts the program execution for the time period specified by (numerical data). The unit is 1/1000 second.

Example:

```
10  WAIT  100
```

..... Interrupts the program execution for 0.1 (100/1000) second.

SEARCH (search)

Searches the character data through the text.

Format:

SEARCH [/P] <character data>

Abbreviated form:

SE.

Explanation:

- Searches for and finds the character data specified in the <character data> through the program text and displays it on the screen.
- When the space key is pressed during execution, the display stops temporarily; when the key is pressed once again, the execution is started again.
- The execution is stopped by SHIFT + BREAK .
- ~~Upper-case and lower-case letters cannot be discriminated within <character data>. Thus, "abc" and "ABC" are considered to be the same character data.~~
- If double quotation marks are used, CHRS (34) is used as the <character data>.
- When /P is specified, data are printed by the printer, not displayed on the screen.

Example:

```
SEARCH "ABC"
```

..... Searches for and finds, in the program text, the line including the character data "ABC", and displays it on the screen.

```
SEARCH "PRINT"+CHR$(34)+"A"+CHR$(34)
```

..... Searches for and finds the line including the character data PRINT "A".

INIT (Initialize)

Sets the RS mode.

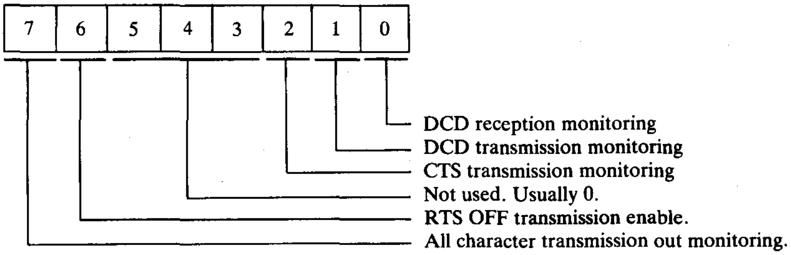
Format:

INIT "RS $\begin{Bmatrix}1\\2\end{Bmatrix}$: <monitoring code>, <initialization setting code>[,<end code>] "

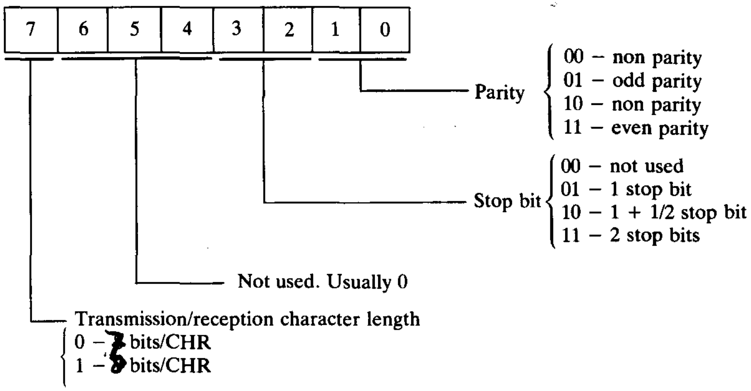
Explanation:

- Sets the mode of RS-232C

<Monitoring code>


7 6 5 4 3 2 1 0
DCD reception monitoring
DCD transmission monitoring
CTS transmission monitoring
Not used. Usually 0.
RTS OFF transmission enable.
All character transmission out monitoring.

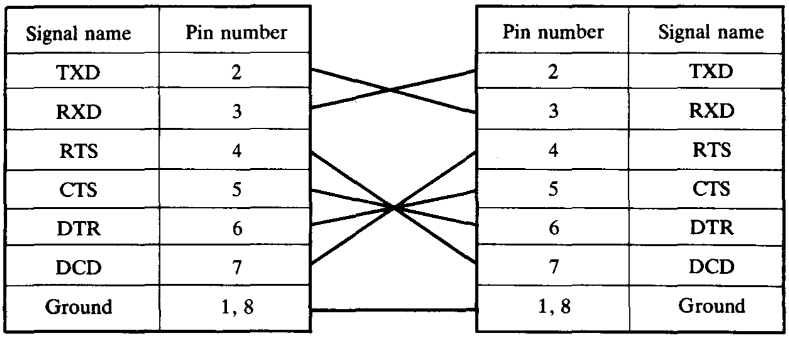
<Initialization setting code>



<ending code>
Values from 0 ~ 255 ($00 ~ $FF)

<breference>

To exchange information between two MZ-700s, prepare a cable provided with the following connections:



Signal name	Pin number
TXD	2
RXD	3
RTS	4
CTS	5
DTR	6
DCD	7
Ground	1, 8

Pin number	Signal name
2	TXD
3	RXD
4	RTS
5	CTS
6	DTR
7	DCD
1, 8	Ground

- Both are used in the terminal mode. (Refer to RS-232C user's manual.)
- The following are programs to transmit/receive numbers from 0 to 9, the contents of A$, when exchanging information between two MZ-700s.

<Reception side>

```
10 INIT "RS1:$00,$8C"
20 ROPEN #1,"RS1:DATA"
30 INPUT #1,A$
40 PRINT A$
50 CLOSE #1
60 END
```

<transmission side>

```
10 INIT "RS1:$00,$8C"
20 A$ = "0123456789"
30 WOPEN #1,"RS1:"
40 PRINT #1,A$    ,
50 CLOSE #2
60 END
```

USR (user)

Calls out and executes in BASIC the machine language program.

Format:

USR (<address> ,<input character variable> , <output character variable)

<address> : numerical data or 4-digit hexadecimal number

Abbreviated form:

U.

Explanation:

- Calls out and executes the machine language program during BASIC program execution. This is the same as the branch command and CALL <address> to the machine language subroutine. Consequently, when there is a return command in a machine language program, control moves to the next statement following the executed statement.
- At the point in time when the machine language program is called out, the value following the <input character variable> is set in the register.
  DE register: heading address of memory area of <input character variable>
  B register: length of <input character variable>
  IX register: address if error-processing routine is announced
- At the point in time of return from the machine language program, the value of the data indicated by the subsequent register becomes the <output character variable>.
  DE register: heading address of memory area of <output character variable>
  B register: length of <output character variable>
- If error-processing is necessary in a machine language program, the following process occurs.
  (1) An error-processing routine is established by the ON ERROR GOTO statement in the BASIC program.
  (2) An error code is substituted in the A register, and is jumped to the address indicated by the IX register.

INP@ (input)

Inputs data at I/O ports to variables.

Format:

INP@ <port number>, <variable>

Abbreviated form:

I.@

Explanation:

- Inputs 8-bit data from <port number> input port and sets to <variable> the value (0 ~ 255) converted to a decimal number.
- From 0 to 127 ($00 ~ $7F in hexadecimal), the <port number> can be determined freely. From 128 to 255 ($F0 ~ $80 in hexadecimal), the use is exclusively as ports for external devices.

Example:

```
10  FOR I =0  TO  20
20  C=I +32
30  GOSUB  "SUB"
40  NEXT  I
50  END
60  LABEL  "SUB"
70  INP@ $FE, A
80  IF  NOT  (A  AND  $0D) =0  THEN  70
90  OUT@ $FF, C
100  OUT@ $FE, $80
110  INP@ $FE, A
120  IF  NOT  (A  AND  $0D) =1  THEN  110
130  OUT@  FE, 0
140  RETURN
```

Reference:

OUT@ (page 64)

OUT@ (OUT)

Outputs data to I/O ports.

Format:

OUT @ <port number>, <numerical data>

Abbreviated form:

OU.@

Explanation:

- Converts <numerical data> values (0 ~ 255) to binary numbers and outputs to the <port number> output port.
- From 0 to 127 ($00 ~ $7F in hexadecimal), the <port number> can be determined freely. From 128 to 255 ($F0 ~ $80 in hexadecimal), the use is exclusively as ports for external devices.
- For control of peripheral devices, etc., data are output to I/O ports. Thus, if there is a mistake of the <port number>, etc. by this OUT@ command, there is the possibility that such a mistake will cause abnormal operation of the peripheral equipment, etc., so care must be taken in that regard.

Example:

```
10 FOR I=0 TO 20
20 C=I+32
30 GOSUB "SUB"
40 NEXT I
50 END
60 LABEL "SUB"
70 INP@ $FE,A
80 IF NOT (A AND $0D)=0 THEN 70
90 OUT@ $FF,C
100 OUT@ $FE,$80
110 INP@ $FE,A
120 IF NOT (A AND $0D)=1 THEN 110
130 OUT@ $FE,0
140 RETURN
```

Reference:

INP@ (page 63)

■ Logical operation

Logical operation

The task of giving YES or ON results, by judgment, are not few in computers. Here let us consider some logical expressions used to judge some conditions. The logical operators NOT, AND, OR and XOR are used in logical expressions.

① X AND Y (logical product = and)
AND means X moreover Y

X Y	X and Y
1 1	1
1 0	0
0 1	0
0 0	0

② X OR Y (logical addition = inclusive or)
OR means either X or Y

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

③ NOT X (negation = not)
Means that it is not X

X	NOT X
1	0
0	1

④ X XOR Y (Exclusive logical addition = exclusive or)
Means X and Y are not equal

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

The logical operation is usually used in IF ~ THEN ~ ELSE command, explained before. For instance, it is used as follows:

- IF −30<X AND Y>20 THEN 120 ELSE 100

  (The meaning above is: if the X value is −30<X moreover Y>29 go to line number 120, if not jump to line 100)
- IF −30<X OR Y>20 THEN 10 ELSE 100

  (The meaning above is: if the X value is −30<X or Y>20 to to line number 120, if not jump to line 100).

Section 3
BASIC MONITOR FUNCTIONS

In order to make the input of machine language programs easier, the DISK BASIC has a monitor section following the IOCS section. As the stack work area, \$FF00 ~ \$FFFF (hexadecimal) are used.
This monitor, in the same way as the BASIC, has a built-in screen editor, and, by using the editing format described below, any address of the main memory 64K bytes can be rewritten.

3.1 EDITING FORMAT

: Address = data␣data␣data

: (Colon)

..... a symbol which indicates an editable line

Address

..... designated by a four digit hexadecimal; main memory address, including heading data (0000 ~ FFFF)

= (equal mark)

..... a separator used to distinguish address and data

Data

..... designated by two digit hexadecimal or semi-colon + character; 8-bit data or specified character ASCII code is written into specified memory address; as a rule, a space is used for data intervals.

3.2 PRINTER SWITCH (P Command)

***P**

Directs D and F commands to printer or screen. When in monitor, becomes screen mode. The mode reverses each time this command is executed. If a printer is not connected, ERR? appears and a command is awaited, so check the printer or execute the P command to return to the screen mode.

3.3 DUMP (D Command)

∗D <heading address[< final address>]>

Displays the memory contents. When the final address is omitted, 128 bytes from the heading address are displayed. When the heading address is omitted, 128 bytes from the subsequent address are displayed. Dumping is by the following format.

$$:\ \underset{\substack{\uparrow\\ \text{Heading address}}}{H}HHH=\underbrace{HH\sqcup HH\sqcup HH\ \ HH\ \ HH\ \ HH\ \ HH\ \ HH}_{\text{8-byte hexadecimal data}}\ \ /\ \underbrace{ABCDE.\,G.}_{\text{8-byte character data}}$$

If the memory content is changed, move the cursor to the data to be changed, make the correction, and then input $\boxed{\text{CR}}$.

Note:
The final 8-byte characters are displayed by data ASCII code, and the control code is indicated by a " . " (period). The display is stopped only by the $\boxed{\text{BREAK}}$ key by $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$ there is a return to awaiting a command.

3.4 MEMORY SET (M Command)

∗M [heading address]

Rewrites the memory content. When the heading address is omitted, rewrite is from the current pointer. To get out of this mode, press $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$.
Because the address and data are displayed and the cursor superimposes over the data, designate the data by the editing format and press $\boxed{\text{CR}}$. The designated data and address will be added and continue to the next line.

3.5 FIND (F Command)

∗F <heading address>␣<final address>␣<data>␣[<data ...>]

Searches from the heading address to the final address for continuous data of the number specified by the data, and, if found, outputs that address and data at the dump mode. Stopped by SHIFT + BREAK.

3.6 GOSUB (G Command)

∗G <call address>

Sub-routine calls the designated call address. Stack pointer is at FFFE (hexadecimal).

3.7 TRANSFER (T Command)

∗T <heading address>␣<final address>␣<transfer heading address>

Transfers data between designated addresses from the transfer heading address.

3.8 SAVE (S Command)

**∗S <heading address>␣<final address>␣<execution heading address>:
[<device name> : filename]**

Records data between designated addresses to the designated device name. The "execution heading address" is the execution heading address when loaded from IPL. Specify the filename in the same way as for DISK BASIC (after a colon " : ").

3.9 LOAD (L Command)

∗L <load heading address> [<device name> : filename]

Loads the designated file from the designated device name. When the heading address is specified, loading is from that address; if not specified, loading of information is done exactly as it was saved. If filename is not specified, the first file found is loaded. When there is BREAK or if check sum error appears, ERR? is displayed, and returns to awaiting command; if no error appears, returns to awaiting command.

3.10 VERIFY (V Command)

∗ V <filename>

Reads designated file from cassette and compares with main memory. Used to check whether correctly saved; if not correct, ERR? is displayed.

3.11 RETURN (R Command)

∗R

Returns monitor to called system. If from BYE command of DISK BASIC, returns to DISK BASIC by this command. SP (stack pointer) and HL register are stored, so next command after BYE is executed.
If SP is called from a system such as $FF00 ~ $FFFF, or if there is no return address in the stack, return is not possible by this command; call that system's hot start by the G command.

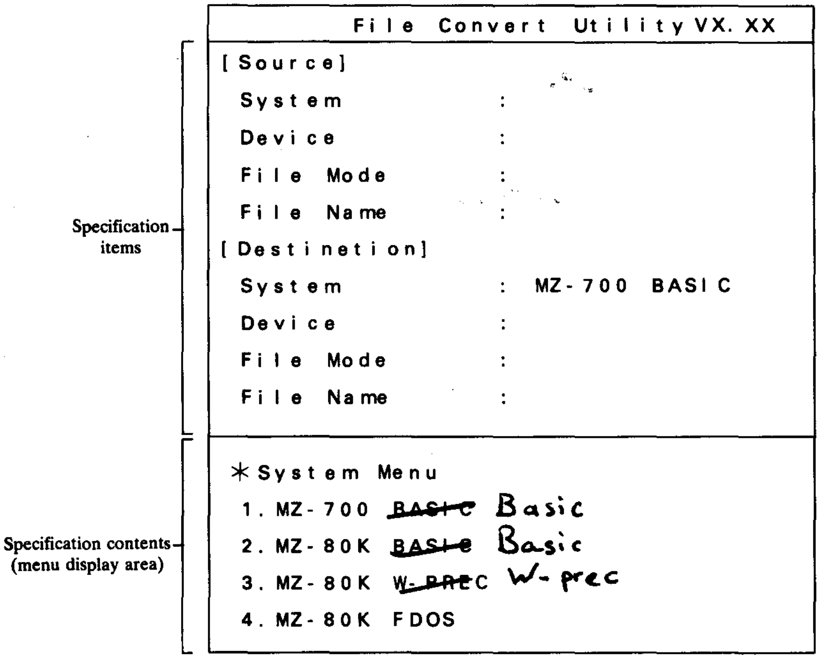
Section 4
APPLICATION PROGRAMS

Use of the File Converter

- The file converter is registered on the DISK BASIC floppy disk by the "TRANS" file name. To execute this, press the following keys (RUN: refer to page 36)

RUN "TRANS" $\boxed{\text{CR}}$

The specification items which appear on the initial screen are specified in order from the top. Choose the right item number from the menu area specification contents and make the key input. The specification item and the initial screen become as follows.



```
          File  Convert  Utility VX. XX
[Source]
  System              :
  Device              :
  File  Mode          :
  File  Name          :
[Destinetion]
  System              :  MZ-700  BASIC
  Device              :
  File  Mode          :
  File  Name          :

 * System  Menu
  1. MZ-700  ~~BASIC~~ Basic
  2. MZ-80K  ~~BASIC~~ Basic
  3. MZ-80K  W-~~PREC~~ W-prec
  4. MZ-80K  FDOS
```

■ Explanation of contents and specification method of menu area display

⋆ System menu

- Specify the source system (the destination system is fixed in the MZ-700). The source program specification inputs: Which MZ series machine was the program made for? For instance, if the program file converter is for the MZ-80K, and moreover if it is made by double precision the DISK BASIC (W-PREC), key input “3”.

- Moreover, the objects are the following systems

MZ-700 Basic	CASSETE TAPE BASIC DISK BASIC
MZ-80K Basic	~~SP-5030~~ ~~SP-6010~~ SP.6015
MZ-80K	~~W-PREC~~ W-Prec
MZ-80K FODS	FDOS S~~YSTEM PROGRAM~~

Display contents

Menu display area

```
*[System Menu]
1. MZ-700 Basic
2. MZ-80K Basic
3. MZ-80K W-Prec
4. MZ-80K FDOS
```

★ **Device menu**

- Specifies the device name (source device, destination device) to perform conversion.

Display contents

Menu display area

```
*[Device menu]
1. FD1:                    5. CMT :
2. FD2:
3. FD3:
4. FD4:
```

★ **File mode**

- Displays the file mode conversion possibilities specified in the system menu and device. However, the destination file mode is made automatically possible file mode from the specification of source file mode.

Display contents

Menu display area

```
*[File Mode]
1. OBJ
2. BTX
3. BSD
4. BRD
```

★ **File name**

- Inputs the file name of file to be conterted (maximum 16 characters). To display the disk directory from this condition, key input $\boxed{\text{CTRL}}$ + $\boxed{\text{A}}$.

Display contents

Menu display area

```
        [File name] input?
  Control-A ==> Directory
```

Display mode

Example of display when CTRL + A is key input and device "1" (FD) is specified.

refer to page 3,4

```
                    File Convert Utility
-----------------------------------------------------------
        DIRECTORY OF FD:
          OBJ "DATA1"
          Function and, ↓ Any key.
          Next page, ↓ Any key.
                    Error message
             Convert again, [Y/N] ?
```

⋆ **Related to the message**

Function and, ↓ Any key.	Returns to the filename input waiting condition when key input (any key) is performed.
Next page, ↓ Any key.	Display when one screen is not enough to display the directory. The display of the remaining directory is performed when once again a key is input.
Error ~~xx~~, ↓ ~~Any Ke~~y. .... Message Convert again ? Y/N.	Indicates that ,because of some error ,directory display is not possible . ~~ot be~~ ~~eturns~~ ~~when~~ When key input is made once again ,awaiting source system input ;when "N" key input ,DISK BASIC is started .

Reference:

If, for file conversion, the source drive and destination drive are the same, the following display is added.

Display:

Set ~~Insert~~ source, ↓ Any key ▩

or

Set ~~Insert~~ destination, ↓Any key ▩

refer to page 4.

Explanation:
The above display occurs if the input of the destination file has ended. First, there will be a display meaning "insert the source disk", and then, after the source disk is inserted into the disk drive, press a key (any key). When the key input occurs, the content of the source disk file is read into the memory of the MZ-700. When this read-in is finished, there will be a display meaning "insert the destination disk". Take out the source disk and insert the destination disk into the floppy disk drive in its place. After insertion is completed, press a key (any key). When the key input occurs, the data in the memory of the MZ-700 will be written into the destination disk.

Display when conversion finishes

- The following display will be shown after the conversion finishes.

```
Message  →   **End of job**
              Break              one of these
              Error message
         Convert again [Y/N]
```

Explanation:
The "END OF JOB" message means that the conversion has finished. "BREAK" means that there was an interruption $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$ during the job.
"ERROR xx" means that there was an error during the job.

After the end of either message display, there will be a display meaning "will conversion be made again? or will conversion be ended?". If conversion will be made again, input "Y", if the end, input "N". When "Y" is input, there will be a return to the initial screen used for convert execution, and, when "N" is input, there will be a return to the monitor condition.

Section 5
DISK BASIC SUMMARY

5.1 Summary of DISK BASIC Commands, Statements, Functions and Operations

5.1.1 Commands

DIR	DIR	Displays floppy disk directory. Information indicated at directory display is as follows: Registered filemode, registration condition (locked or not) and filename Note: For the directory display on the CRT screen, there is a stop when one screen amount is displayed, and the cursor appears. To continue the directory display the CR key can be pushed, or it is possible to move to another command.
	DIR FD1	Displays the directory of the floppy disk in floppy disk drive number 1. When the DIR command is executed, the system memorizes that drive number, and thereafter the drive number can be omitted to designate direct execution commands and file access commands for that same floppy disk drive.
	DIR	Displays the directory of the floppy disk in the floppy disk drive that was executed by the most recent DIR command.
DIR/P	DIR/P " FD1"	Prints floppy directory on line printer.
LOAD	LOAD "FD1:DAY"	Reads out BASIC text (BTX) with "DAY" file name in floppy disk.
	LOAD "FD2:SUN", A	Consider BSD file "SUN" filename in floppy disk drive number 2 as ASCII format and read it out.
	LIMIT $D000: LOAD "B"	For read-out of machine language program file (OBJ) linked to a BASIC text, it is necessary to separate the machine language area and the BASIC area by the LIMIT command. Refer to the command used for linking with the machine language program.

SAVE	SAVE "FD1:DAY"	Names the BASIC text currently in the text area "DAY" and writes in to floppy disk. One file with filename "DAY" and file mode BTX is registered.
	SAVE "CMT:E"	Names the BASIC text currently in the text area "E" and writes in to the cassette tape.
RUN	RUN	Executes the program from the heading of the BASIC text currently in the text area. Note: At the RUN command, all variables become 0 or null immediately prior to program execution.
	RUN 1000	Executes program from statement number 1000.
	RUN "FD1:F"	Reads out BASIC program file "F" from floppy disk, and executes program from the program heading.
	RUN "FD3:G"	Reads out program text "G" from volume number 7 of the floppy disk in drive number 3, and then executes program from the designated execution address. In this instance, the system is not controlled by BASIC.
MERGE	MERGE	Adds the program in the file to the program.
	MERGE "FD1:PR OG"	Merges program currently in the memory and "PROG" file in the floppy disk.
VERIFY	VERIFY "H"	Compares program text currently in BASIC text area and content of cassette tape file specified by file name "H".
AUTO	AUTO	Automatically generates line numbers 10, 20, 30 ... during text making.
	AUTO 200, 20	Automatically generates 200, 220, 240 ... in steps of 20, from statement number 200. AUTO command is released by pressing SHIFT + BREAK keys.
LIST	LIST	Displays all lists of BASIC text currently in text area.
	LIST-500	Displays list up to statement number 500.
LIST/P	LIST/P	Display list goes to printer. (TEXT MODE)

RENUM	RENUM	Changes statement number of the program.
	RENUM 100	Renumbers all statements beginning with first statement number 100, and in steps of 10.
SEARCH	SEARCH "ABC"	Searches for and finds lines including "ABC" character data in program text, and displays on screen.
NEW	NEW	Erases BASIC text currently in text area and clears variable area. Machine language area specified by LIMIT command is not cleared.
CONT	CONT	Continues program execution. In other words, restarts execution from point of interruption by SHIFT + BREAK keys or STOP statement during program. CONT command becomes invalid when, during a program break, the BASIC text is edited.
BYE	BYE	Moves system control from DISK BASIC to monitor. (The return from monitor to DISK BASIC can be made by monitor command "R".)
KEY LIST	KEY LIST	Lists, on the CRT display, the definition condition of the definable function keys.

5.1.2 File Control Statements

LOCK	LOCK "FD2:ABC"	Locks file "ABC" in floppy disk drive number 2. Locked files cannot be changed or deleted. In the directory display, locked files are denoted by the * symbol.
UNLOCK	UNLOCK "ABC"	Unlocks "ABC" file in active drive.
	100 UNLOCK "FD1: A"	Executes the program unlocking of "A" file in floppy disk drive number 1.

RENAME	RENAME "FD1:A,B"	Changes filename of file "A" in floppy disk drive number 1 to filename "B".
DELETE	DELETE "A"	Deletes file "A" from disk in default drive.
CHAIN	CHAIN "FD1:TEXT B"	Chains program execution to BASIC text "TEXT B" in floppy disk. In other words, "TEXT B" is read out to BASIC text area, and program execution continues from that heading. The original program in the text area at this time is made NEW, and the content of the variable and user function is transferred to the chained text. It can be understood that the function of the CHAIN statement is as a GOTO "filename".
SWAP	SWAP "FD2:TEXT S-R"	Swaps program execution to BASIC text "TEXT S-R" in floppy disk drive number 2. In other words, the text in execution is once shunted to the floppy disk in the default drive, then "TEXT S-R" is read out to the BASIC text area, and program execution continues from that heading. When the swapped program is finished, the original text is then read out, and the program execution continues from the next statement after the SWAP statement. When each program execution is linked, the content of the variable and user function is transferred. The SWAP level must not exceed 1. In other words, SWAP instructions cannot be made within a swapped text. In can be understood that the function of the SWAP statement is as the GOSUB "filename".

5.1.3 BSD (BASIC Sequential Access, Data File) Control Statements

Note: For file descriptors FD and CMT

WOPEN #	WOPEN #3 "FD2:SEQ DATA1"	Opens the file for write-in so that one BASIC sequential access file (BSD) can be made. In other words, it defines the filename of the BSD being made as "SEQ DATA1", and opens, as logical number 3, a file in floppy disk drive number 2.
PRINT #	PRINT #3, A, A$	Writes in, in order, the content of variable A and string variable A$ on BSD; a file opened in logical number 3 by the WOPEN # statement. File close is executed by the CLOSE # statement, and BSD is formally registered as one BSD.
CLOSE #	CLOSE #3 (Corresponding to WOPEN #)	Closes the BSD, the file opened in logical number 3 by the WOPEN # statement. By closing the file, one BSD with the file name specified by the WOPEN # statement is made on the specified floppy disk, and the logical number (3 in this instance) becomes undefined once more.
KILL #	KILL #3	Kills the BSD, the file opened in logical number 3 by the WOPEN # statement. In other words, it cancels or erases the BSD, and the logical number (3 in this instance) becomes undefined once more.
ROPEN #	ROPEN #4, "FD2:SEQ DATA2"	Opens the file for read-out of the data in the BASIC sequential access data file (BSD). In other words, it opens, as logical number 4, the BSD file "SEQ DATA2" in volume number 7 of the floppy disk in floppy disk drive number 2.
INPUT #	INPUT #4, A(1), B$	Reads out sequential data from the BSD, a file opened in logical number 4 by the ROPEN # statement, and substitutes numerical data in array variable A(1) and string in string variable B$. Read-in data are sequentially accessed from the BSD heading data.

CLOSE #	CLOSE #4 (Corresponding to ROPEN #)	Closes the BSD; the file opened in logical number 4 by the ROPEN # statement. Logical number 4 becomes undefined once more.

5.1.4 BRD (BASIC Random Access, Data File) Control Statements

Note: For file descriptor FD only.

XOPEN #	XOPEN #5 "FD3:DATA R1"	Opens (cross opens) for data write-in/read-out to the BASIC random access file (BRD). In other words, cross opens BRD file "DATA R1" on the floppy disk in drive number 3 to logical number 5, or, if the file does not yet exist, cross opens so that the BRD file "DATA R1" can be newly made on that floppy disk.
PRINT #( )	PRINT #5(11), R(11)	Writes in the content of a one-dimensional numerical array variable R(11) to element 11 of the BRD file opened in logical number 5 by the XOPEN # statement.
	PRINT #5(20), AR$, AS$	Writes in string variables AR$ and AS$ to, respectively, elements 20 and 21 of the same BRD as above. Because all elements in the BRD are a fixed length of 32 bytes, any part of the string length which exceeds 32 bytes will be invalid.
INPUT #( )	INPUT #5(21), R$	Reads out (substitutes), to string variable R$, the data in element 21 of the BRD file opened to logical number 5 by the XOPEN # statement.
	INPUT #5(11), A(11), A$(12)	Reads out the data in elements 11 and 12 of the BRD, as described above, to the one-dimensional numerical array variable A(11) and one-dimensional string array variable A$(12) respectively.
CLOSE #	CLOSE #5	Closes the BRD file opened to logical number 5 by the XOPEN # statement.
	CLOSE	Closes all files opened by WOPEN, ROPEN or XOPEN.

KILL #	KILL #5	Kills the BRD opened in logical number 5 by the XOPEN # statement.
	KILL	Kills all files opened by WOPEN, ROPEN or XOPEN.
IF EOF (#)	IF EOF (#5) THEN 700	If the file end occurs during execution of the INPUT # statement relative to BSD, or during execution of the INPUT # ( ) statement relative to BRD, this is a branch statement that commands to jump to the processing routine in statement number 700.

5.1.5 Error Processing Statements

ON ERROR GO TO	ON ERROR GOTO 1000	If an error occurs during program execution, this is a sentence saying to jump to statement number 1000.
IF ERN	IF ERN = 44 THEN 1050	If the error number is 44, this is a command to jump to statement number 1050.
IF ERL	IF ERL = 350 THEN 1090	A command to jump to statement number 1090 if the error statement number is 350.
	IF (ERN = 53) ✳ (ERL = 700) THEN END	A command to finish the program if the error number is 53 and the error statement number is 700. For the DISK BASIC, if an error occurs during the program, the error number and error statement number will be set, respectively, to variables ERN and ERL.
RESUME	650 RESUME	Transfers control once again to the command generating the error.
	700 RESUME NEXT	Transfers control to the command following the command generating the error.
	750 RESUME 400	Transfers control to statement number 400.
	800 RESUME 0	Transfers control to the program heading.

5.1.6 Substitution Statements

LET	LET A=X+3	Substitutes sum results of numerical variable X and numerical data 3 to numerical variable A. LET can be omitted.

5.1.7 Input/Output and Color Control Statements

COLOR	10 COLOR , , , 2	Changes all screen background color to red.
	20 COLOR 3,2,7	Changes the color of characters at coordinates (3,2) to white.
	30 COLOR 4,2,4,2	Makes the color of characters at coordinates (4,2) green, and the background color red.
PRINT	10 PRINT A	Displays the content of numerical variable A on the CRT display.
	?A$	Displays the content of string variable A$ on the CRT display.
	100 PRINT [6,5] "ABC"	Writes the "ABC" string in yellow on a light blue background.
	110 PRINT [,4] "DEF"	Writes the "DEF" string in yellow on a green background.
	120 PRINT [7,4] "GHI"	Writes the "GHI" string in white on a green background.
	200 PRINT	New line if PRINT only.
PRINT USING	PRINT USING "###.##";A	A designation which lines up decimal point positions by a fixed decimal point display.
INPUT	10 INPUT A	Inputs values relative to variable A from the keyboard.
	20 INPUT A$	Inputs strings relative to string variable A$ from the keyboard.
	30 INPUT "VALUE?";D	Before input from the keyboard, the question string data VALUE? is displayed. The semi-colon is used to separate the string from the variable.

	40 INPUT X, X$, Y, Y$	Numerical variables and string variables can be combined by using the comma (,) to separate them, but it is necessary to match the type of variable at the time of input.
SET	SET 30,15	Illuminates the position of coordinates (30,15).
RESET	RESET 30,15	Erases the position of coordinates (30,15).
GET	10 GET N	Inputs one numerical character from the keyboard relative to numerical variable N. If the key is not pressed at that time, 0 is input.
	20 GET K$	Inputs one string from the keyboard relative to string variable K$. If the key is not pressed at that time, A$ becomes vacant.
READ ~ DATA	10 READ A,B,C 1010 DATA 25, −0.5, 500	Numerical data 25, −0.5 and 500 are substituted to, respectively, numerical variables A, B and C by execution of the READ-DATA statements at the left.
	10 READ H$, H, S$, S 30 DATA "HEART", 3 35 DATA "SPADE", 11	The first data of the DATA statement, i.e., string data "HEART", is substituted for the first variable of the READ statement, i.e., for the string variable H$. Next, numerical data 3 is substituted for the second variable H, and read-in continues one after the other.
RESTORE	10 READ A,B,C 20 RESTORE 30 READ D,E 100 DATA 3, 6, 9, 12, 15	In the example at the left, 3, 6 and 9 are respectively substituted for variables A, B and C by the READ statement in statement number 10, but, because the RESTORE statement occurs next, the values next substituted for variables D and E by statement number 30's READ are, respectively, 3 and 6, not 12 and 15.
	700 RESTORE 200	Moves the data read-out pointer in the READ-DATA statement to the heading of the DATA statement in statement number 200.

5.1.8 Loop statements

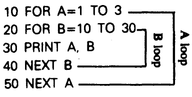
FOR ~ TO NEXT

```
10 FOR A = 1 TO
   10
20 PRINT A
30 NEXT A
```

The statement number 10 is a command to change variable A and substitute for values from 1 to 10; the value of the first A becomes 1. Because the value of A is displayed on the CRT screen by statement number 20, the numeral 1 is displayed. Next, the value of A becomes 2 by statement number 30, and this loop is repeated. The loop is repeated in this way until the value of A becomes 10. (At the point when the loop ends, the value 11 is entered to A.)

```
10 FOR B = 2 TO
   8 STEP 3
20 PRINT B
30 NEXT
```

A command to change variable B and substitute for values from 2 to 8 in steps of 3 (statement number 10). It is also possible to make the STEP value negative and make the variable smaller each time.



```
10 FOR A=1 TO 3
20 FOR B=10 TO 30
30 PRINT A, B
40 NEXT B
50 NEXT A
60 NEXT B, A
70 NEXT A, B
     ↑
Becomes error.
```

An example of an overlay of the FOR ~ NEXT loops (variables A and B). Note that B loop is placed inside A loop. Nesting of loops (doubling, tripling ...) is possible, but the inner loop must be enclosed within the outer loop. FOR ~ NEXT nesting must not exceed 15 levels.
It is possible, by the previous double loop, to group statement numbers 40 and 50, as in statement number 60 at the left, into one NEXT statement. For an operand such as shown in statement number 70, however, an error occurs.

5.1.9 Branch Statements

GOTO	100 GOTO 200	Jumps to statement number 200 (= movement of program execution).
GOSUB ~ RETURN	100 GOSUB 700 .... 800 RETURN	Branches to statement number 700 subroutine (calling of subroutine). Ends subroutine execution by RETURN statement, and returns to statement following GOSUB command in the main program.
IF ~ THEN	10 IF A>20 THEN 200	Jumps to statement number 200 if variable A is larger than 20. Executes next statement if A is 20 or less.
	50 IF B<3 THEN B =B+3	Substitutes B + 3 for variable B if variable B is less than 3. Executes next statement if B is 3 or greater.
IF ~ GOTO	100 IF A>=B GO TO 10	Jumps to statement number 10 if variable A is equal to or greater than variable B. Executes next statement if A is less than B.
IF ~ GOSUB	30 IF A=B * 2 GO SUB 90	Branches to statement number 90 subroutine if value of variable A is equal to twice the value of B. If not, executes next statement. (If there is a multi-statement following a conditional statement, the ON statement is executed when the condition is not reached, but the IF statement moves the execution to the next statement number if the condition is not reached, and the multi-statement is ignored.)
ON ~ GOTO	50 ON A GOTO 70, 80, 90	Jumps to statement number 70 if variable A is 1, to statement number 80 if it is 2, and to statement number 90 if it is 3. The next statement is executed if A is 0 or 4 or more. The INT function is included in ON, so jumps to statement number 80 if A is 2.7, in the same way as 2.
ON ~ GOSUB	90 ON A GOSUB 700, 800	Branches to statement number 700 subroutine if variable A is 1, and to statement number 800 if it is 2. The next statement is executed if A is 0 or 3 or more.

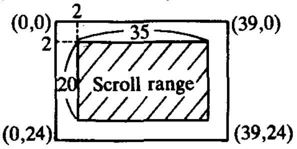
5.1.10 Definition Statements

DIM	10 DIM A(20)	For one-dimensional numerical array variable A( ), 21 array variables become available, from A(0) to A(20).
	20 DIM B(79,79)	For two-dimensional numerical array variable B( ), 6400 array variables become available, from B(0, 0) to B(79, 79).
	30 DIM C1$(10)	For one-dimensional string array variable C1$( ), 11 array variables become available, from C1$(0) to C1$(10).
	40 DIM K$(7,5)	For two-dimensional string array variable K$( ), 48 array variables become available, from K$(0, 0) to K$(7, 5).
DEFAULT	DEFAULT "FD1"	Considered to be floppy disk drive number 1 if device names are omitted by commands.
DEF FN	100 DEF FNA(X) = X↑ 2 − X 110 DEF FNB(X) = LOG(X) + 1 120 DEF FNZ(Y) = LN(Y)	Statement number 100 defines $X^2 - X$ to FNA(X), statement number 110 defines $\log_{10}X + 1$ to FNB(X), and statement number 120 defines $\log_e Y$ to FNZ(Y). Each function is limited to 1 variable.

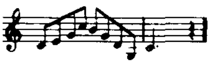
DEF KEY	15 DEF KEY(1) = "L IST" + CHR$(13) 25 DEF KEY(2) = " L OAD:RUN" + CHR$ (13)	The DEF KEY statement of statement number 15 defines the function LIST $\boxed{\text{CR}}$ to function key number 1, and statement number 25 defines the function LOAD:RUN $\boxed{\text{CR}}$.
INIT	INIT "RS1:"	Sets the RS-232C mode.
LABEL	210 LABEL "SUB" 220	Defines subroutine name beginning from statement number 210 as "SUB".

5.1.11 Comment Statements and Control Statements

REM	200 REM JOB-1	REM is a comment statement; ignored when program is executed.
STOP	850 STOP	Stops program execution and awaits command. If CONT command given here, program continues.
END	2000 END	Indicates end of program. Executes program end.
CLR	300 CLR	All numerical variables and character variables become 0 or vacant (null); all array variables return to undetermined condition. All DEF FN statements also become invalid.
CLS	10 CLS	Erases the secreen in the scroll range.
CURSOR	50 CURSOR 25, 15 60 PRINT "ABC"	Specifies the position by numerals or variables: from 0 ~ 39 from the left end in the X axis direction, and 0 ~ 24 from the top end in the Y axis direction. For the example at the left, string "ABC" is displayed from the 26th cursor position from the left end of the screen and the 16th cursor position from the top end.

CONSOLE	10 CONSOLE 0, 25, 0, 40	The scroll range covers the whole screen.
	20 CONSOLE 5, 15	Specifies the scroll range from the 5th line to the 10th line.
	30 CONSOLE 0, 25, 5, 30	Specifies the scroll range from the 5th line to the 30th line.
	40 CONSOLE 0, 10, 0, 10	Specifies the scroll range to a 10×10 range.
	50 CONSOLE 2, 20, 2, 35	Specifies the scroll range to the scroll range shown in the figure below.  (0,0) 2 (39,0) 35 2 20 Scroll range (0,24) (39,24)
SIZE	? SIZE	Displays the unused size (in bytes) of the BASIC text area.
TI$	100 TI$ = "222030"	Sets the internal clock to 10:20:30 PM. Time data are expressed as a 6-digit figure within quotation marks.

5.1.12 Music Control Statements

MUSIC **TEMPO**	300 TEMPO 7 310 MUSIC "DE#FGA"	Tempo 7 (fastest speed) is specified by statement number 300. By statement number 310, re mi ~~fa#~~ fa sol la (mid-range) are played at tempo 7. If there is no TEMPO statement, the music is played at the tempo of the default value.
	300 M1$ = "C3EG + C" 310 M2$ = "BGD − G" 320 M3$ = "C8R5" 330 MUSIC M1$,M2$,M3$	In this example, the melody is substituted to the 3 string variables and the MUSIC command is executed. When the staff notation is used, the notes below are played. Note that, because there is no TEMPO statement, the playing is at the default value tempo. 

5.1.13 Machine Language Program Control Statements

INP@	INP@ $E8,A	Substitutes data at port number $E8 for variable A.
OUT@	OUT@ $E8,A	Outputs variable A to port number $E8.
LIMIT	100 LIMIT 49151	Limits the area used by the BASIC program to the 49151 address (BFFF with hexadecimal notation).
	100 LIMIT A	Limits the area used by the BASIC program to the address of variable A.
	100 LIMIT $BFFF	Limits the area used by the BASIC program to the address BFFF in hexadecimal notation. A hexadecimal notation is indicated by an "$" mark before the notation.
	300 LIMIT MAX	Returns the area used by the BASIC program to the maximum memory.
POKE	120 POKE 49450, 175	Sets data 175 (decimal notation) to the decimal notation address 49450.
	130 POKE AD, DA	Sets the value (0 ~ 255) indicated by variable DA to the address specified by variable AD.
PEEK	150 A = PEEK (494 50)	Changes the data at decimal notation address 49450 to a decimal number, and substitutes for variable A.
	160 B = PEEK(C)	Changes data entered at the decimal notation address specified by variable C to a decimal notation, and substitutes for variable B.
USR	500 USR(49152)	Moves program control to decimal address 49152. This control movement has the same function as the machine language CALL command. As a result, when the RET command (201 at decimal notation) is in the machine language program, returns to the BASIC program.
	550 USR (AD)	Calls the decimal address specified by variable AD.
	570 USR($C000)	Calls the hexadecimal address C000.

5.1.14 Printer Control Statements

AXIS		Valid in GRAPH mode.
	30 AXIS 0, −10, 48	Adds a scale of 48 graduations in increments of 10 to the Y-coordinate axis from the current pen position.
	50 AXIS 1, 10, 48	Adds a scale of 48 graduations in increments of 10 to the X-coordinate axis from the current pen position.
CIRCLE		Valid in GRAPH mode
	50 CIRCLE 0, 0, 240, 0, 360, 0, 2	Draws a circle (radius 240) from coordinates (0,0).
GPRINT		Valid in GRAPH mode.
	30 GPRINT (2,2), "A"	Prints the character A upside down at the size of the 26 digit mode of the TEXT mode.
HSET		Specifies the current pen position to a new starting point. (Valid in GRAPH mode.)
LINE		Valid in GRAPH mode.
	10 LINE% 1, 240, 0, 240, −240, 0, −240,0, 0	Coordinates (240,0), (240,−240), (0,−240) and (0,0) are connected by a solid line from the current pen position.
MODE	MODE TN	Returns from the graphic mode to the text mode (40 characters per line).
	MODE TL	Returns from the graphic mode to the text mode (26 characters per line).
	MODE TS	Returns from the graphic mode to the text mode (80 characters per line).
	MODE GR	Switches from the text mode to the graph mode (in order to draw graphs and figures).
MOVE		Valid in GRAPH mode.
	10 MOVE 150, 100	Moves the pen upward from the current pen position to coordinates (150, 100).

RMOVE		Valid in GRAPH mode.
	20 RMOVE −240, 240	Moves the pen upward relatively from the current pen position by −240 (X direction) and 240 (Y direction).
PAGE		Valid in TEXT mode.
	10 PAGE 30	Specifies 30 lines per page.
PCOLOR		Valid in both TEXT and GRAPH mode
	10 PCOLOR 1 20 PR INT/P "ABC"	Prints "ABC" to the plotter printer in blue.
PHOME		Moves the pen upward from the current pen position and returns to the starting point. (Valid in GRAPH mode)
PLOT	PLOT ON	Enables use of color plotter printer as substitution for the display. (Valid in TEXT mode.)
	PLOT OFF	Cancels above function.
PRINT/P		Valid in TEXT mode
	10 PRINT/P A,A$	Outputs string variable A$ content after the numerical variable A content to printer.
	20 PRINT/P "[illegible]"	For form feed of printer.
PRINT/P USING		Outputs format specified data to screen. Format specification is written after the word USING. ( )
	PRINT/P USING "####";A	Numerical variable A contents are output to printer within 4 digits, justified right.
RLINE		Valid in GRAPH mode.
	70 RLINE% 1, 240, 0, −120, −SQ, −12, SQ	Connects specified positions, relatively from current pen position (240,0), (−120,−SQ) and (−120,SQ) by solid line.
SKIP		Valid in TEXT mode.
	10 SKIP 10	Advances the paper 10 lines.
	20 SKIP −10	Rewinds 10 lines.
TEST		Checks color specification and ink amount and dryness. (Valid in TEXT mode)

LN	100 A = LN (X)	Regarding the value of variable X, gets natural logarithm $\log_e X$ and substitutes for variable A. X must be a positive value.
	110 A = LOG (X)/LOG(Y) 120 A = LN (X)/LN (Y)	In order to obtain $\log_Y X$ when the logarithm base is Y, it can be obtained by statement number 110 or statement number 120.
RND	100 A=RND (1) 110 B=RND (10)	When there is a positive integral number in parentheses, such as in statement number 100 or 110, generates a random number using values from 0.00000001 to 0.99999999 sequentially each time the RND function is used. (This has no relationship to the positive integral number in parentheses.)
	200 A=RND (0) 210 B=RND (−3)	When there is a 0 or negative integral number in parentheses, such as in statement number 200 or 210, initialization of random number generation occurs, a specific number is always generated, and the same value is substituted for A and for B.

5.1.15 String Control Functions

LEFT$	10 A$ = LEFT$ (X$, N)	Substitutes string variable X$ (from beginning to Nth character) for string variable A$. It doesn't matter whether N is a constant, variable or numerical formula.
MID$	20 B$ = MID$ (X$, M, N)	Substitutes string variable X$ (from Mth character to N character) for string variable B$.
RIGHT$	30 C$ = RIGHT$ (X $, N)	Substitutes string variable X$ (from end to N character) for string variable C$.
SPC	40 D$ = SPC (N)	Substitutes N number of spaces for string variable D$.
CHR$	60 F$ = CHR$ (A)	Converse to the ASC function, substitutes ASCII code characters which are equivalent to the value of real number A for string variable F$. It doesn't matter whether A is a constant, variable or numerical formula.
ASC	70 A = ASC (X$)	Substitutes the value of the ASCII code of the first character of string variable X$ for variable A.
STR$	80 N$ = STR$ (I)	Converse to the VAL variable, substitutes the numerical variable I as if it were a string for string variable N$.
VAL	90 I = VAL (N$)	Substitutes the numerical string of string variable N$ as if it were a number for variable I.
LEN	100 LX = LEN (X$)	Substitutes the character length (character number) of string variable X$ for variable LX.
	110 LS = LEN (X$ + Y$)	Substitutes the sum of the character length of string variables X$ and Y$ for variable LS.

5.1.16 Tab Function

TAB	10 PRINT TAB (X);A	Displays the value of variable A at the X + 1 character position counting from the left edge of the screen.

5.1.17 Arithmetic Operations

The calculation priority is of white figures on dark background at left side, but the calculation of figures in parentheses ( ) has even higher priority.

↑	10 A = X ↑ Y (power)	Substitutes the X ↑ Y calculation result for variable A. (Note, however, that an error occurs if Y is not an integral number when X is a negative number at X ↑ Y.)
−	10 A = −B (minus sign)	0 − B is a subtraction; note that the "−" of −B is a minus sign.
✻	10 A = X ✻ Y (multiplication)	Substitutes the multiplication result of X and Y for variable A.
/	10 A = X/Y (division)	Substitutes the division result of X and Y for variable A.
+	10 A = X + Y (addition)	Substitutes the addition result of X and Y for variable A.
−	10 A = X − Y (subtraction)	Substitutes the subtraction result of X and Y for variable A.

5.1.18 Comparison Logic Operators

=	10 IF A = X THEN	If variables A and X are equal, executes commands from THEN onward.
	20 IF A$ = "XYZ" THEN	If string variable A$ content is string XYZ, executes commands from THEN onward.
>	10 IF A>X THEN ...	If variable A is greater than X, executes commands from THEN onward.

<	10 IF A<X THEN ...	If variable A is smaller than X, executes commands from THEN onward.
<> or ><	10 IF A<>F THEN ...	If variable A and X are not equal, executes commands from THEN onward.
>=or=<	10 IF A>=X THEN	If variable A is greater than or equal to X, executes commands from THEN onward.
<=or=<	10 IF A<=X THEN	If variable A is smaller than or equal to X, executes commands from THEN onward.
*	40 IF (A>X) * (B> Y) THEN	If variable A is greater than X and variable B is greater than Y, executes commands from THEN onward.
+	50 IF (A>X) + (B> Y) THEN	If variable A is greater than X or variable B is greater than Y, executes commands from THEN onward.

5.1.19 Other Symbols

?	200 ? "A+B="; A +B 210 PRINT "A+B ="; A+B	Can be used instead of PRINT. Consequently, statement number 200 and 210 are the same.
:	220 A=X:B=X ↑2: ?A,B	A symbol to express punctuation of the command statement; used in multiple commands. There are 3 command statements used in the statement number 220 multiple command.
;	230 PRINT "AB"; "C D"; "EF"	Executes PRINT continuously. As a result line number 230, "ABCDEF" is displayed on the screen continuously, with no space.
	240 INPUT "X="; X $	Displays "X =" on screen; awaits data key input of string variable X$.
,	250 PRINT "AB","C D","E"	Executes PRINT with tabulation. For statement number 250, first AB is displayed on the screen, then CD is displayed in the position 10 characters to the right of A, and then E is displayed in the position 10 characters to the right of C.

	300 DIM A(20), B$ (3,6)	An example used in punctuation of a variable.
" "	320 A$ = "DISK BASIC" 330 B$ = "MZ-700"	" " indicates a string content
$	340 C$ = "ABC" + CHR$(3)	Indicates a string variable.
	500 LIMIT $BFFF	Indicates hexadecimal number.
π	550 S = SIN (X ✱ π/180)	The approximate value of pi (3.1415927) is expressed by π.

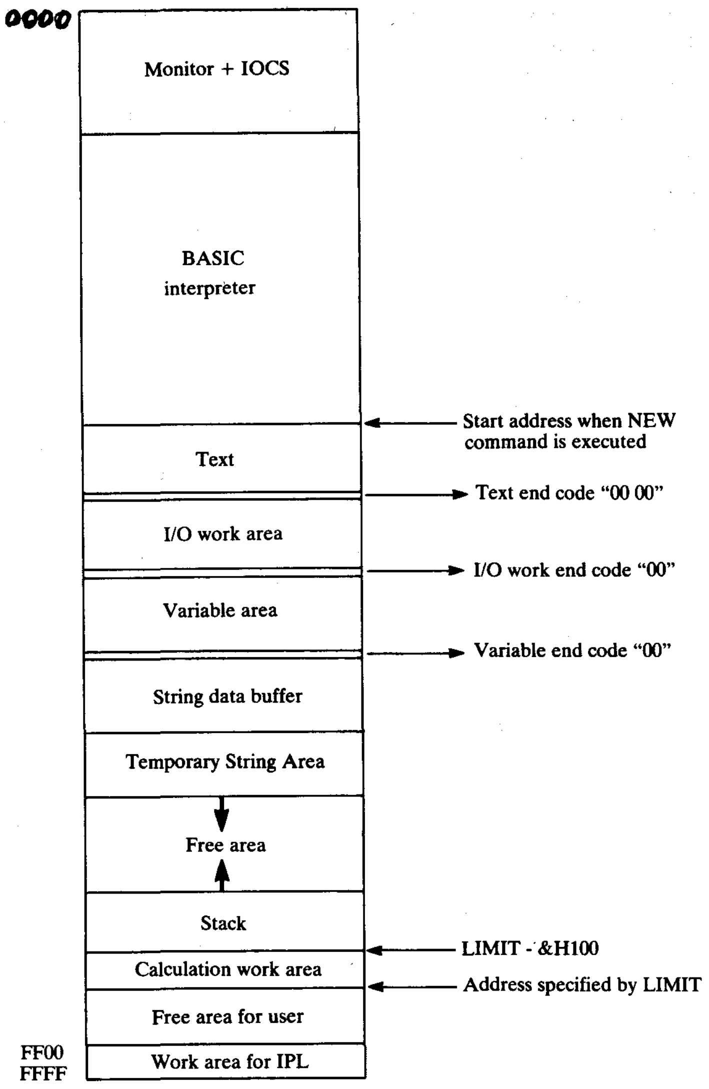
5.2 Error Message List (DISK BASIC)

Error display number	Statement	Content
1	Syntax error	Error in syntax
2	Overflow error	Numerical data outside the range, operation result overflows.
3	Illegal data error	Illegal number or variable used.
4	Type mismatch error	The data type and variable type do not match.
5	String length error	String length exceeds 255
6	Memory capacity error	Insufficient memory capacity
7	Array def. error	Attempt to define same array variables larger than before using the undefined array variables.
8	LINE length error	Length of one line exceeds the limit.
10	GOSUB nesting error	GOSUB statement nesting over
11	FOR — NEXT nesting error	FOR — NEXT statement nesting over 15
12	DEF FN nesting error	Function definition (by DEF FN statement) nesting over 6
13	NEXT error	NEXT statement without FOR statement
14	RETURN error	RETURN statement without GOSUB statement
15	Un def. function error	Use of undefined function
16	Un def. ~~line num~~. error	Attempt to refer to an undefined statement number
17	Can't ~~continue~~ cont error	Continuation of program impossible by CONT statement
18	Memory protection error	Request for write-in in BASIC interpreter control area
19	'Instruction error	Use of direct command and statement mixed
20	Can't resume error	RESUME can't be executed

Error display number	Statement	Content
21	Resume error	Attempt to use RESUME although no error
24	Read error	Use of READ statement without corresponding DATA statement
25	SWAP ~~level~~ error	Swap level exceeds 1
28	System id error	Trying to access file other than DISK BASIC.
29	Framing error	Framing error
30	Overrum error	Overrum error
31	Parity error	Parity error
40	File not found error	Reference non-existent file
41	Disk drive hardware error	Floppy disk drive hardware error
42	Already exist ~~Same~~ file error	Attempt to register already existing filename
43	Already open error	Opening of file already open
44	~~File~~ not open error	Referencing (or CLOSE or KILL) unopen file
46	Write protect ~~file~~ error	Write-in prohibited file
50	~~Disk~~ not ready error	Floppy disk off as system
51	Too many files error	Attempt to register beyond maximum number 63 of files
52	Disk mismatch error	
53	No file space error	Insufficient space on floppy disk
54	~~File~~ unformat error	Non-initialized floppy disk
55	~~BSD data overflow~~ Too long file error	Data size to 1 BSD file exceeds 64K bytes
58	Dev. name error	Error in device name description
59	Can't execute error	Trying to execute device that cannot be executed.
60	Illegal filename error	Filename error

Error display number	Statement	Content
61	Illegal file mode error	File mode error
63	Out of file error	Out of file (file data read-in)
64	Logical number error	Non-regulation logical number
65	LPT: not ready	Printer not connected
68	Dev. mode error	Error in device mode
69	Unprintable error	Non-defined error message
70	Check sum error	Check sum error (tape read-in error)

5.3 Memory Map



5.4 Use of the Floppy Disk

For information regarding the method used for setting the floppy disk and the method of floppy disk drive operation, please refer to the Operation Manual for model MZ-IF02.

It should be remembered that a master disk should be handled and used very carefully. Moreover, because Sharp optional floppy disks are not initialized, please initialize them by a utility program before use.

Notes regarding the use of floppy disks are not initialized; please initialize them by a utility program before use.

Notes regarding the use of floppy disks

■ If fingerprints get on the floppy disk through the head window, read-out and write-in will become impossible. Take the utmost care not to allow the surface of the floppy disk to become marked, stained, dirtied or defaced in any way!

■ Storage temperature conditions (around the floppy disk): 4°C ~ 53°C (39°F ~ 127°F)
The jacket will become deformed if the temperature exceeds 53°C (127°F). Please take care not to expose the floppy disk to direct sunlight for a long time and not to place it in an environment where the temperature is apt to exceed 53°C (127°F). When the floppy disk is used, please use it within the temperature range prescribed on the protection sleeve. Moreover, because the environmental conditions of the place of use differ from those of the place of storage, it is suggested that, before use, the floppy disk be placed for a short time in conditions corresponding to those of the place of use.

■ When inserting the floppy disk into the floppy disk drive, insert it straight, gently, and all the way until it stops. Then move the front door lever to the horizontal position. Rough handling will damage the floppy disk.

■ Do not bend or fold the floppy disk, If the jacket becomes deformed, read-out and write-in will become impossible.

■ Write information on the index label before attaching it to the jacket. If something must be written on a label which is already attached, use a felt marking pen or similar soft-tip instrument; do not use a pencil or ball-point pen.

■ Smoking, eating or drinking in the vicinaty of the floppy disk drive or floppy disks should be discouraged, or should be done only if great care is taken that ashes, food particles or liquids do not get into the floppy disk drive or on the floppy disks.

Notes regarding the storage of floppy disks

■ Absolutely avoid placement or storage near sources of magnetism. Data on disks can be erased by magnetized rings, necklaces, etc., so the wearing of any item which might possibly be magnetized should be carefully avoided when floppy disks are handled. Note also that it is also dangerous to bring floppy disks near other equipment or devices which generate or emit magnetism. Note, for example, that computer CRT displays, cassette recorders and household television sets all generate magnetism, so floppy disks should always be kept away from these and similar equipment.

■ Be sure to store the floppy disk in its protective sleeve when it is not being used. The habit should be formed to immediately insert the floppy disk in the sleeve when it is removed from the floppy disk drive. By following this practice, the majority of handling errors and accidents can be prevented. Special care should be taken regarding the master disk: place it in the floppy disk drive only when it is to be used, and when it is not being used it should be immediately and carefully stored in a safe place.
The protective sleeve is made of a special material in order to prevent damage by static electricity and moisture, so be sure to always store the floppy disk only in this sleeve.

■ For storage, insert the floppy disk in its protective sleeve and then place it in the storage box. The box should be placed so that the disks inside stand vertically. Avoid any storage conditions in which the disk is not standing vertically straight or in which it is bent or apt to become bent.

A storage box for the master disk is not included. To store it, use a box which is applicable, keeping in mind the conditions stated above.

- Avoid holding floppy disks with pager clips, spring clips or any other similar instrument.

- Never place any heavy item on top of a floppy disk. Also do not carelessly place them on a desk top, etc. Be sure to always return all floppy disks to their prescribed place for storage immediately after use.

Printed in Japan
Gedruckt in Japan
Imprime' au Japon
Stampato in Giappon

4C 0.1101

TINSE1140ACZZ

SHARP CORPORATION
OSAKA, JAPAN