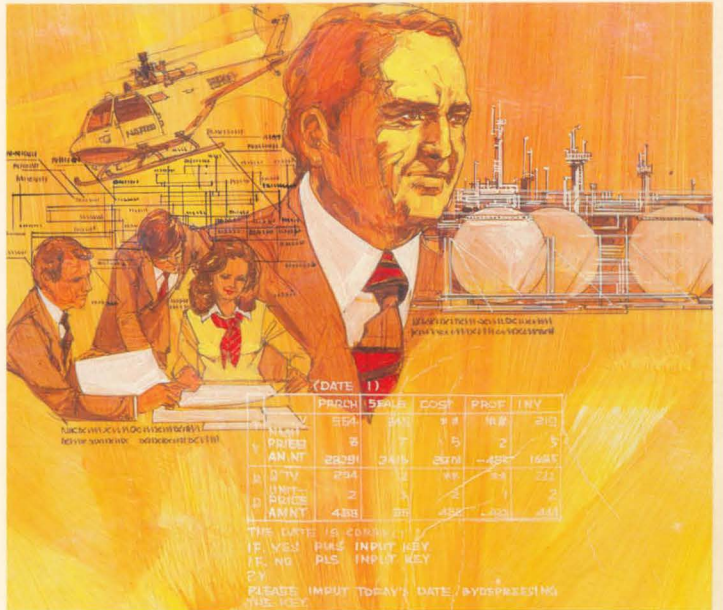
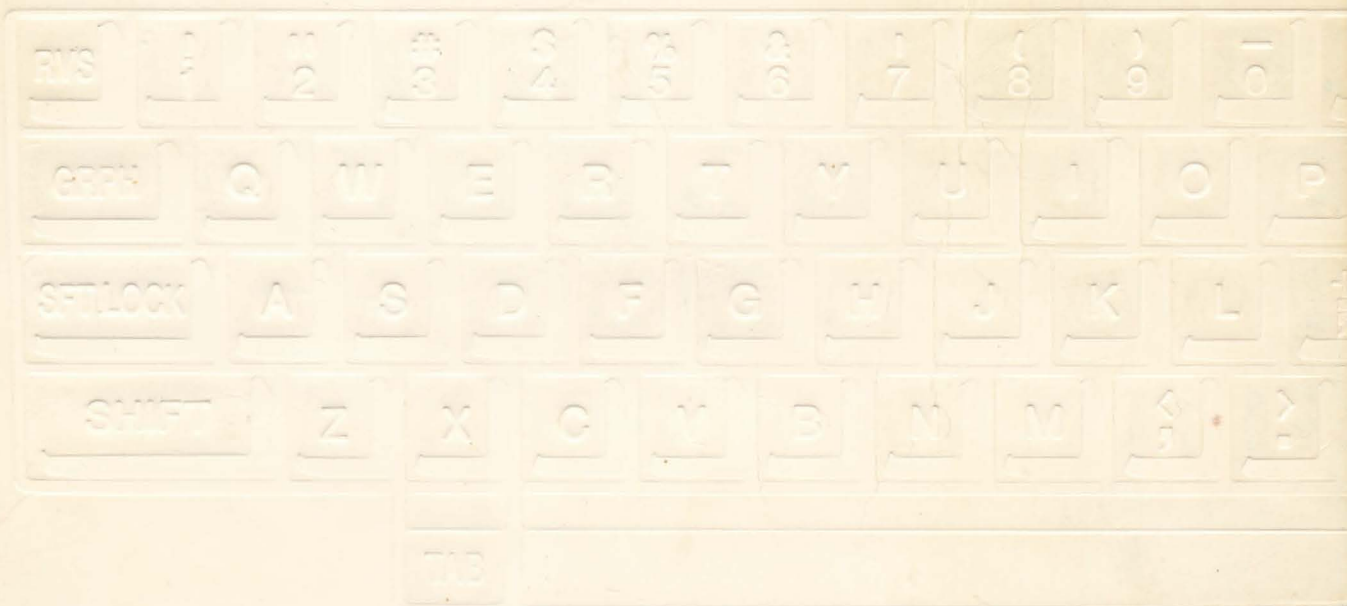


Personal Computer MZ-80B

DISK BASIC MANUAL



F1 F2 F3 F4 F5 F6 F7 F8 F9 F10



SHARP

SHARP

Personal Computer

MZ-80B

DISK BASIC Manual

January 1981

080211-150281

NOTICE

This manual is applicable to the SB-6510 DISK BASIC interpreter used with the SHARP MZ-80B Personal Computer. The MZ-80B general-purpose personal computer is supported by system software which is filed in software packs (cassette tapes or diskettes).

All system software is subject to revision without prior notice, therefore, you are requested to pay special attention to file version numbers.

This manual has been carefully prepared and checked for completeness, accuracy and clarity. However, in the event that you should notice any errors or ambiguities, please feel free to contact your local Sharp representative for clarification.

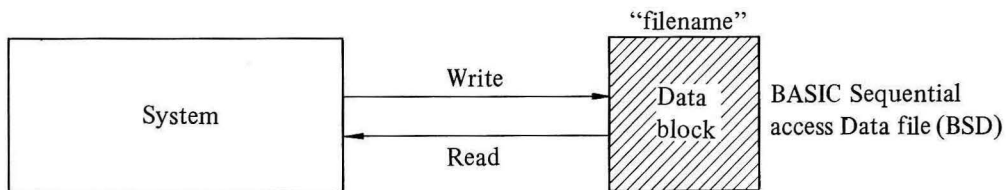
All system software packs provided for the MZ-80B are original products, and all rights are reserved. No portion of any system software pack may be copied without approval of the Sharp Corporation.

Introduction

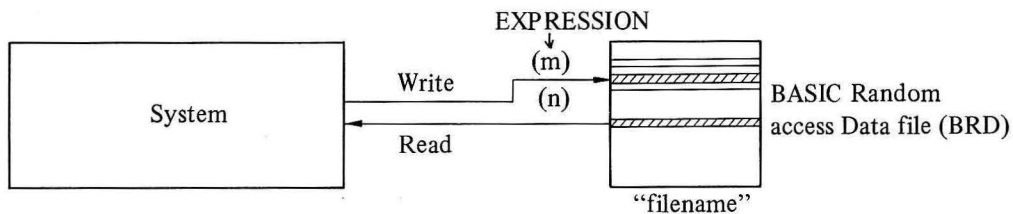
The DISK BASIC interpreter SB-6510 is a system software which has a superb file control function. It fully utilizes the large capacity and high speed accessing feature of the floppy disk file system so that files can be used not only for data storage but also as a random access data area connected to the system program. Further, with this interpreter disk files can be used as program segments which may be called for execution in job units by the program in memory with the CHAIN or SWAP statements.

Data files are classified into two groups according to the file access method: sequential access files and random access files.

A sequential access file is a block of file data which can be accessed sequentially. Data are accessed sequentially from the beginning by specifying the file name.



A random access file is a set of file data which can be accessed at random. Each data item is written in the file as an array element and is assigned with an expression with which the system controls it.



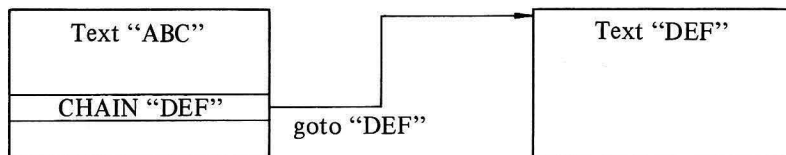
In general, when data can be treated in segments (e.g., decimal data used when coding a program by POKE statement) or it is arranged according to a certain rule (e.g., elements of a table), it is effective to write it as a sequential access file. When particular data items need to be accessed (e.g., in the case of information retrieval), it is effective to write it as a random access file.

To access data, first specify the file (a set of data assigned a file name) with a logical file number of 1 to 127. A logical file number is assigned to a file with a logical open statement as an alternative to the file name.

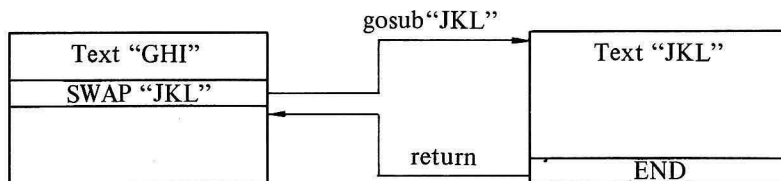
The file to which the specified logical file number has been assigned is accessed by the write or read command issued by a PRINT # or INPUT # statement or by a file close statement.

CHAIN and SWAP are statements which overlay a program upon another program in the memory and transfer control to the overlying program.

The CHAIN statement is used as a GOTO "filename" function.



The SWAP statement is used as a GOSUB "filename" function. Control will be returned to the first program after the overlaid program has been completed. (In this case, overlay is performed again.)



This manual is written with the assumption that readers are familiar with the ordinary BASIC language.

The greatest care must be taken in handling disk drives and diskettes. Carefully read the notes in the Floppy Disk Operator's Guide. Notes on handling diskettes are included in Appendix 4.

The master diskette and blank diskette will not be exchanged for new ones after purchase. It is recommended that the master diskette be copied using the disk copy utility to generate a submaster diskette, and that the submaster diskette be used ordinarily. Be sure to keep the master diskette in a safe place.

Contents

Notice	<i>ii</i>
Introduction	<i>iii</i>
Chapter 1 Outline of DISK BASIC SB-6510	1
1.1 Activating the DISK BASIC interpreter SB-6510	2
1.2 Introduction to data file control	3
1.3 Control of sequential access files	4
1.4 Control of random access files	7
1.5 Making a chain of programs	10
1.6 Swapping programs	11
1.7 Reserved word	13
1.8 Initialization	14
Chapter 2 Instructions Unique to SB-6510	15
2.1 Commands	16
2.1.1 DIR	16
2.1.2 DIR/P	16
2.1.3 SAVE	17
2.1.4 SAVE/T	17
2.1.5 VERIFY	17
2.1.6 LOAD	18
2.1.7 LOAD/T	18
2.1.8 RUN	18
2.2 File control statements	19
2.2.1 LOCK	19
2.2.2 UNLOCK	19
2.2.3 RENAME	19
2.2.4 DELETE	20
2.2.5 CHAIN	20
2.2.6 SWAP	21
2.2.7 WOPEN #	21

2.2.8	PRINT #	22
2.2.9	CLOSE #	22
2.2.10	KILL #	22
2.2.11	ROPEN #	23
2.2.12	INPUT #	23
2.2.13	XOPEN #	24
2.2.14	PRINT # ()	24
2.2.15	INPUT # ()	25
2.2.16	IF EOF (#) THEN	25
2.3	Error processing control	26
2.3.1	ON ERROR GOTO	26
2.3.2	IF ERN	26
2.3.3	IF ERL	27
2.3.4	RESUME	28
2.4	Use of utility programs	29
2.4.1	Use of utility program "Filing CMT"	29
2.4.2	Use of utility program "Utility"	30
Chapter 3 Data Processing Application		35
Chapter 4 Programming Instruction		65
4.1	List of DISK BASIC interpreter SB-6510 commands, statements and functions	66
4.1.1	Commands	66
4.1.2	File control statements	68
4.1.3	BSD (BASIC Sequential access Data file) control statements	69
4.1.4	BRD (BASIC Random access Data file) control statements	69
4.1.5	Error processing statements	70
4.1.6	Cassette data file input/output statements	71
4.1.7	Assignment statement	71
4.1.8	Input/output statements	71
4.1.9	Loop statement	73
4.1.10	Branch statements	73
4.1.11	Definition statements	74
4.1.12	Comment and control statements	75
4.1.13	Music control statements	76

4.1.14	Graphic control statements	76
4.1.15	Machine language control statements	78
4.1.16	Printer control statements	79
4.1.17	I/O input/output statements	79
4.1.18	Arithmetic functions	80
4.1.19	String control functions	81
4.1.20	Tabulation function	82
4.1.21	Arithmetic operators	83
4.1.22	Logical operators	83
4.1.23	Other symbols	84
APPENDIX		85
A.1	ASCII Code Table	86
A.2	Error Message Table	88
A.3	Memory Map	90
A.4	Handling diskettes	91
 SUPPLEMENT DISK BASIC Error List & File Commands		

Chapter 1

Outline of DISK BASIC SB-6510

This chapter outlines programming procedures and use of the DISK BASIC interpreter SB-6510. The chapter begins with a description of the procedure for activating the BASIC SB-6510, followed by general file control concepts.

For details of file control statements and use of the utility programs, see Chapter 2.

For other commands, statements, functions, operators and symbols, see Chapter 4.

1.1 Activating the DISK BASIC interpreter SB-6510

DISK BASIC SB-6510 is stored (along with MONITOR SB-1510) on a diskette file and must undergo initial program loading whenever it is to be used. Loading is easily performed. Ready the disk drive unit, place the master diskette (or submaster diskette[†], if available) in disk drive 1 and simply turn on the power of the MZ-80B.

The MZ-80B's built-in IPL (Initial Program Loader) automatically starts (photo at left in Figure 1.1), loading both the DISK BASIC interpreter SB-6510 and the MONITOR SB-1510. When loading is completed, the MZ-80B displays the message illustrated in the photo at right and the DISK basic interpreter begins to operate.

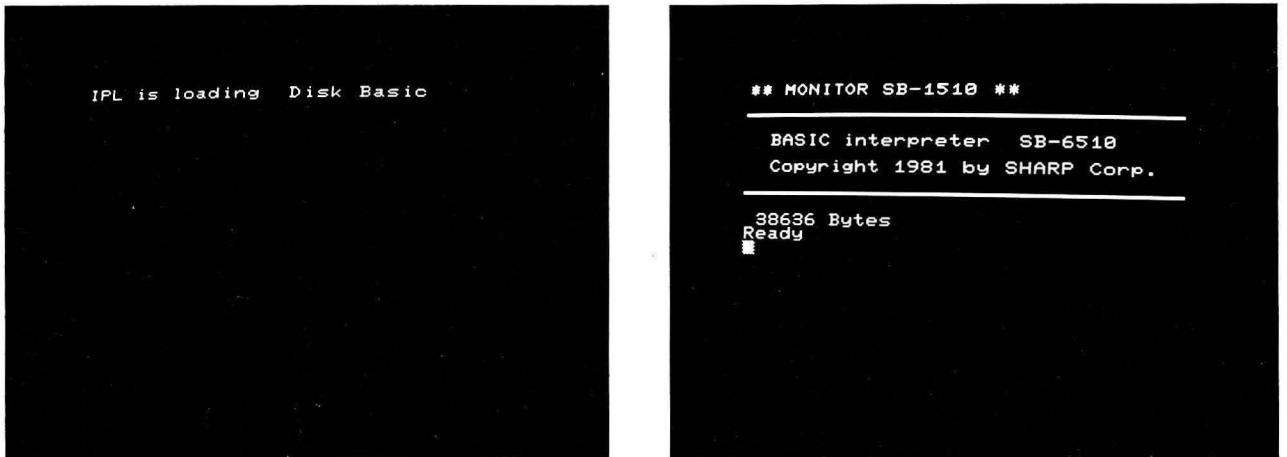


FIGURE 1.1

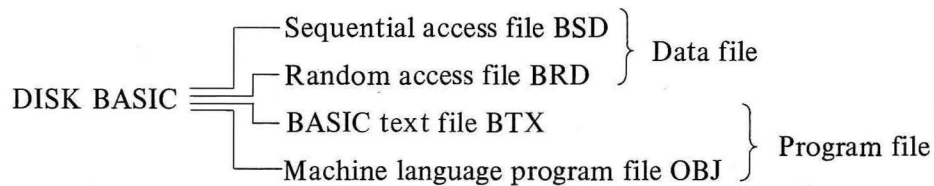
The SB-6510 automatically loads and executes the program assigned the file name "AUTO RUN" which is stored on the master (submaster) diskette. This program defines the functions assigned to the 10 special function keys. By assigning "AUTO RUN" to another program, the program can be automatically loaded and executed after IPL.

[†] Procedures for making a submaster diskette are explained on page 32.

1.2 Introduction to data file control

Now we will discuss handling and control of various data and program files that make use of the large storage capacity and high-speed access function of our floppy disk unit.

We have already noted that DISK BASIC is capable of handling three kinds of files: two data files – the sequential access file (BSD) and random access file (BRD) – plus one program file – the BASIC text (BTX). One more file, the machine language program file (OBJ), has been constructed using a system program or MONITOR SB-1510 and recorded on the master diskette. This file is intended to be run alone or used linked with the program in the BASIC text area; hence, DISK BASIC can utilize it, but cannot write it or change its contents.



In discussing individual file control instructions, we will first explain procedures for constructing and using the two kinds of data files; second, we will explain use of the CHAIN and SWAP file statements.

1.3 Control of sequential access files

A sequential access file is a data file whose data is recorded or read with sequential access procedures, which accesses data sequentially starting with the first data item.

You already know how to handle data files on cassette tape using BASIC SB-5510. Sequential access with DISK BASIC is the same, except that the medium is not a cassette tape but a diskette. The method is, of course, far more practical and provides high speed access, enabling more versatile file control through several new file control statements.

First, let's compare the DISK BASIC and cassette-based BASIC sequential access statements.

Recording files (writing data)

	DISK BASIC	Cassette-based BASIC
File open statement	WOPEN #n, "file name"	WOPEN "file name"
Data write statement	PRINT #n, data	PRINT/T data
File close statement	CLOSE #n	CLOSE
Cancel statement	KILL #n	

Recalling files (reading data)

	DISK BASIC	Cassette-based BASIC
File open statement	ROPEN #n, "file name"	ROPEN "file name"
Data read statement	INPUT #n, variable	INPUT/T variable
File close statement	CLOSE #n	CLOSE
File end detection	IF EOF (#n) THEN	

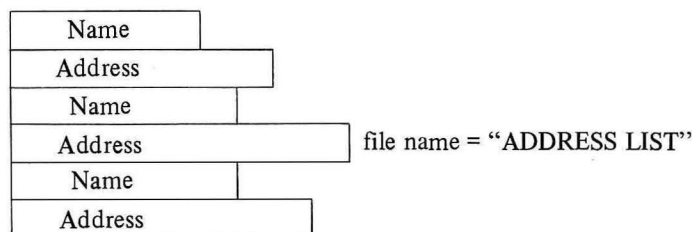
Note: The general form of a file open statement includes specification of the drive number and diskette volume number. These are not shown in the tables above.

As you can see, the statements of these two BASICs closely corresponds to each other in composition. Did you notice that each DISK BASIC statement always contains the symbol "#n"? This is called a logical number, and it must be specified whenever a DISK-BASIC-based file is to be accessed.

Cassette-based BASIC permits either writing to or reading from only one file, whereas DISK BASIC statement utilizes the parallel arrangement of multiple files in the disk system to enable arbitrary access and control of multiple files (**maximum of 10 files**) simultaneously. In addition, it is devised so that files opened can be defined with arbitrarily chosen logical numbers, making it unnecessary to write their file names every time you want to specify them thereafter.

(This difference is a result of differences in hardware cassette and diskette; that is, cassette files are essentially sequential in nature, while disk files are random.)

As a simple example of sequential access file control, let's discuss the recording of names and addresses of persons' homes in a sequential access file. Our file, an address list in this example, must be made in the following form.



The reason the above rectangles are not the same length is that data recorded with the sequential access method is not fixed in length. In a random access file, as we will later mention, all data is given a fixed length of 32 bytes. When all data is handled in the blocks as in this example, or when most of the data (addresses in this example) is too long to be recorded in 32 bytes or is not fixed in length, the sequential access file is more suitable.

Shown below is a program which causes the system to behave as follows: substitute string variables alternately with names and addresses with the INPUT statement, record a combination of names and addresses one by one to make "ADDRESS LIST" with 50 combinations in all, then read stored data out of the file (list) and display it on the CRT screen in groups of 10 items.

(Writing)

```
100 WOPEN #3, FD1@22, "ADDRESS LIST"
110 FOR P = 1 TO 50
120 INPUT "NAME="; NA$
130 INPUT "ADDRESS ="; AD$
140 PRINT #3, NA$, AD$
150 NEXT P
160 CLOSE #3
```

(Reading)

```
200 ROPEN #4, FD1@22, "ADDRESS LIST"
210 FOR P = 1 TO 5: FOR Q = 1 TO 10
220 INPUT #4, NA$, AD$
230 PRINT NA$: PRINT AD$
240 NEXT Q
250 PRINT "STRIKE ANYKEY"
260 GET X$: IF X$ = "" THEN 260
270 NEXT P
```

```
280 PRINT "END"  
290 CLOSE #4
```

Data is stored by this program in slave diskette volume number 22 in drive 1. The meaning of each statement used in this program will be described in the following chapter.

How to detect file end

What is the result when the number of data reads exceeds the number of recorded items? In such cases, no error occurs and the variables are set with 0 or "null", then a special function, EOF (#n), detects file end. EOF (#n), becomes "true" if it comes to the end of a file while data is being read with the INPUT# statement. Hence, if the statement

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

is placed after an INPUT# statement, instructions following THEN are executed when EOF (#n) becomes true (when the file end is detected).

[Problem]

By modifying the sample program, make a program to read and display data from the file "ADDRESS LIST" in groups of ten. The number of records is, however, unknown.

[Sample solution]

A sample solution is as follows.

```
300 ROPEN #5, FD1@22, "ADDRESS LIST"  
310 FOR P = 1 TO 10  
320 INPUT #5, NA$, AD$  
330 IF EOF (#5) THEN 400  
340 PRINT NA$: PRINT AD$  
350 NEXT P  
360 PRINT "STRIKE ANYKEY"  
370 GET X$: IF X$ = " " THEN 370  
380 GOTO 310  
400 CLOSE #5  
410 PRINT "FILE END" : END
```

1.4 Control of random access files

A random access file is a data file which permits data to be recorded or recalled using the “random access” method. The term “**random access**” refers to the process of recording or recalling each data item by specifying it as an array element. Unlike sequential access files, random access files permit addressing any data elements included in a collection of data.

The PRINT# and INPUT# statements used in random access statements contain an “expression” which specifies the array elements following the logical number, as shown below. This is because random access files require designation of the arrays of data of which they are composed.

PRINT #n (expression), data

INPUT #n (expression), variable

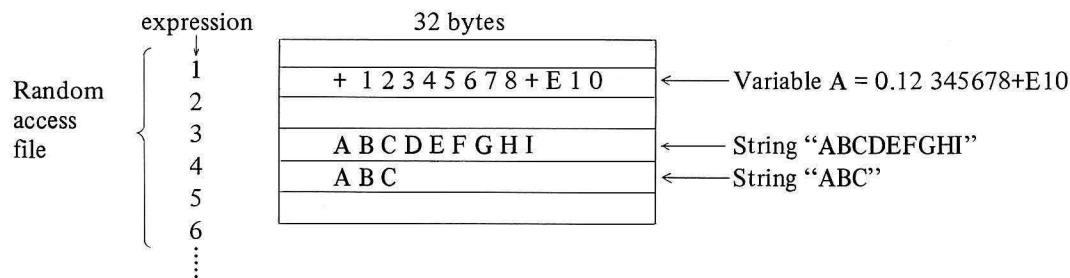
↑
Array element designation

The “**expression**” must be given as a numerical value or variable. The statement

INPUT #7 (21), A\$

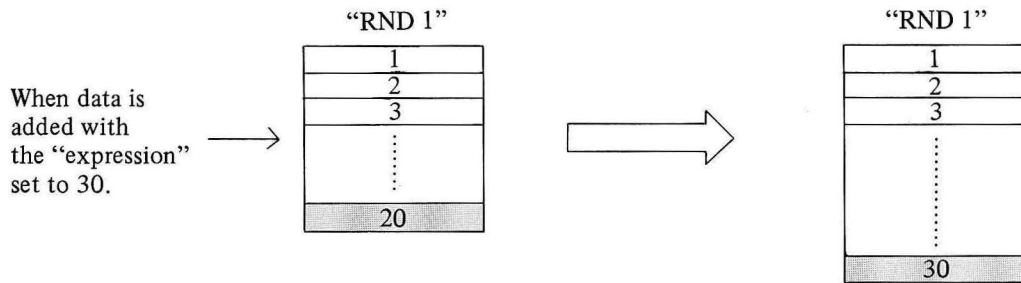
for example, commands the system to read the 21st data element of a group in a random access file opened with logical number #7 into variable A\$.

Note that random data access requires that every data item be recorded in a fixed length. In other words, random access files require recording numeric and string variables in 32 bytes or less.



Numeric variables, including those expressed in exponential notation, do not usually exceed 32 bytes, whereas string variables may extend up to 255 bytes. String variables exceeding 32 bytes cannot be recorded in one data element of a random access file.

Another difference between random and sequential access files is that a random file can be expanded after it has been initially created. Given random access file “RND 1” recorded using an “expression” of 20, for instance, the file may be expanded to accommodate 30 “boxes” when data is newly entered with the “expression” set to 30.



Now, let's try to devise a program for making a simple inventory list using a random access file. It is assumed that individual articles are given fixed item numbers from 1 to 50 and that the inventory list includes five fields of information: item name, unit price, number of units in stock, value (unit price X number items in) and comments.

When recording inventory data for each article, its item number must be entered first.

Recording inventory data

```

100 XOPEN #5, "STORE LIST"
110 INPUT "ITEM NO. ="; K
120 IF K = 0 THEN 300
130 INPUT "ITEM NAME ="; N$
140 INPUT "UNIT PRICE ="; P
150 INPUT "NO. OF UNITS ="; S
160 INPUT "COMMENTS ="; C$
170 T = P * S
180 PRINT #15 (K * 5 - 4), N$, P, S, T, C$
190 GOTO 110
300 CLOSE #5
310 END

```

A random access file made with the above program is as follows. If the item number assigned is $K = 12$, the five kinds of data entered are stored in elements indicated by the expressions corresponding to 56 through 60.

expression	55	
$K * 5 - 4$	}56	N\$: item name
$K = 12$		P: unit price
		S: number of stock
		T: value
		C\$: comments
	61	

BRD file
"STORE LIST"

In this way, data can be arbitrarily arrayed in the file. Hence the file, unlike a sequential access file which is filled with data in succession, may include empty locations, providing for simple data rewriting. Next, let's devise a program to recall the random access file "STORE LIST" made as shown above and display inventory data for a certain article.

Recalling inventory data

```
500 XOPEN #17, "STORE LIST"
510 INPUT "ITEM NO. = "; J: IF J = 0 THEN 700
520 INPUT #17 (J*5-4), N$, P, S, T, C$
530 PRINT "NO."; J: PRINT "ITEM NAME:"; N$
540 PRINT "UNIT PRICE:"; P
550 PRINT "NO. OF UNITS:"; S
560 PRINT "VALUE:"; T
570 PRINT "COMMENTS:"; C$
580 GOTO 510
700 CLOSE #17
710 END
```

In this way, random access files enable the inventory data on specific articles to be called at once by inputting their article numbers, no matter how many articles are inventoried.

1.5 Making a chain of programs

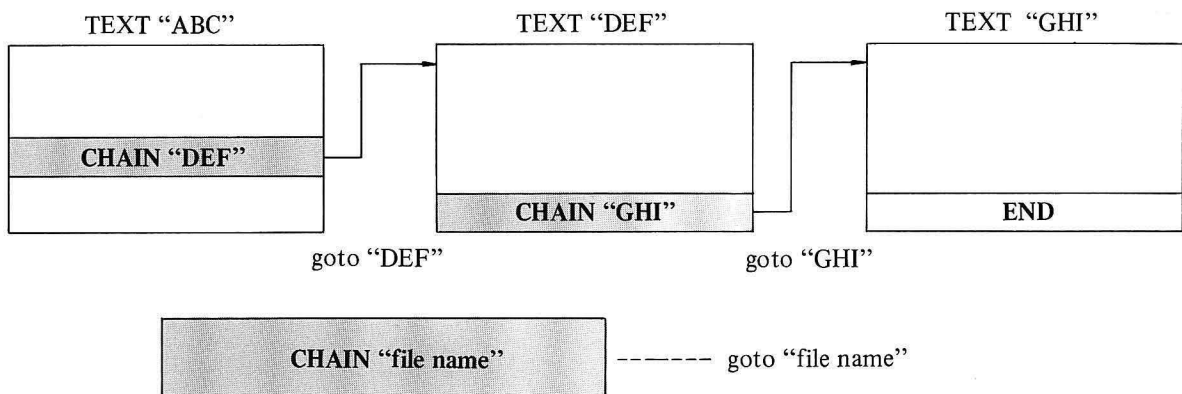
The topic of this section is two program file control statements. These are the CHAIN and SWAP statements. When some programs are recorded on a diskette, the use of these statements enable you to call another program while running the recorded programs and move the control to it. In detail, the CHAIN statement enables you to connect any program to the ones recorded on a diskette, and the SWAP statement enables you to call any program in the form of subroutine. First is described the CHAIN statement to connect or join programs.

The form of the CHAIN statement is as follows.

```
CHAIN FD1@50, "TEXT 2"
```

This statement commands the system to clear a program then present in the text area (it, however, keeps the values of variables), overlay that area with the text named "TEXT 2" that is recorded on the diskette of volume number 50 present in drive 1 and move control to the head of that text. The execution of this text frees the system from the control of the then running BASIC text and compels it to read the text "TEXT 2" anew, moving control to its head. **When two programs are connected, the values of variables and the function defined by the DEF FN in the original program are kept.**

The function of the CHAIN statement can be grasped as one of "GOTO" statement.



The use of the CHAIN statement enables you to process such a huge program as to overflow the BASIC text area by dividing it into pieces and then uniting them again as illustrated above. That is, the CHAIN statement joins component programs every time they are processed. Therefore, the statement and the SWAP statement we will next refer to can be said to be an indispensable aid in coping with complicated, versatile data processing in small businesses.

Apart from such a sophisticated application, it is quite exciting and interesting to join various texts on a diskette. The DISK BASIC, as seen from this, has an original world – which cannot be created by the conventional BASIC – in that enables programs to extend themselves.

1.6 Swapping programs

The SWAP statement reads a program from a diskette file, overlays another program with it or link them, and leaves control to that program text, resuming control by the original program the instant the execution of the text has been completed. Such behaviour is just the same as referring to a subroutine in a text; a fetched program returns to the location next to the one that has been subjected to the SWAP statement. Hence, the SWAP statement can be grasped as a subroutine call. To achieve the above-mentioned action correctly a program text that has the SWAP statement must be temporarily stored in a diskette before the execution of swapping. The program control process cannot then return to the stored original program text before the text area is renewed and the subprogram is called and completely executed. The SWAP statement is generally available in the following form.

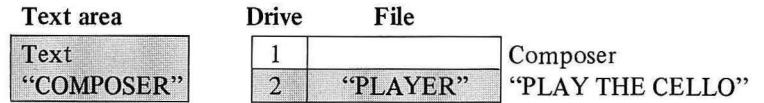
```
SWAP FDd@v, "file name"
```

This form orders the system to swap a subprogram specified by "file name" that is stored on the diskette with volume number v present in drive d (d = 1 to 4). Storing of a program text prior to execution of a subprogram occurs onto the diskette present in the drive that has last executed the DIR FDd command. This means that the drive must be loaded with a diskette that allows temporary writing of a program text. The swapping level must be less than 1.

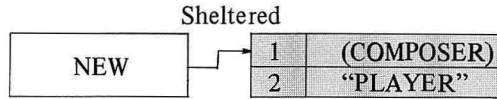
Let's follow the program file behaviour by taking a simple example in order to understand the SWAP statement. How does the file when the DIR FD1 command is executed?

<pre>[Program present in the text area] 10 REM COMPOSER 20 M1\$ = "A7B6 + C3A7A3" 30 M2\$ = "B + C + D + E6A3" 40 M3\$ = "+ F6A3 + E7" 50 PRINT "PLAY THE CELLO" 60 SWAP FD2@7, "PLAYER" 70 PRINT "VERY GOOD" 80 END</pre>	<pre>[Program file "PLAYER"] 10 REM CELLO PLAYER 20 MUSIC M1\$, M2\$, M3\$ 30 PRINT "OK?" 40 END</pre> <p style="text-align: center;">↑ This file is present on slave diskette No. 7 inserted in drive No. 2.</p>
--	---

Initially, the text "COMPOSER", present in the text area, is executed.



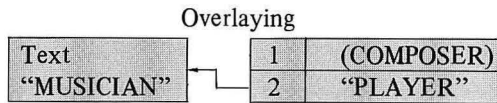
First the SWAP statement, line No. 60, shelters the text on the diskette present in the drive FD1 that has executed the DIR command, and renews the text area.



Player plays melodies.

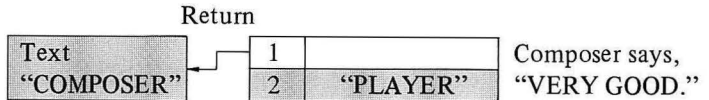


Second the text area is overlaid with BTX "MUSICIAN" and the program is executed to play melodies.



"OK?"

On the completion of playing, the sheltered COMPOSER returns, saying "VERY GOOD."



1.7 Reserved word

A BASIC sentence is composed of reserved words—also called key words—which include statements, built-in functions and special signs (and also commands), and other elements, such as constants, variables, arrays and expressions. Table 1.1 shows all reserved words of the DISK BASIC interpreter SB-6510.

[A]	ABS		IMAGE/P		REM
	ASC		INP		RENAME
	ATN		INPUT		RESET
	AUTO		INPUT#		RESTORE
[B]	BLINE		INPUT/T		RESUME
	BOOT		INT		RETURN
[C]	CHAIN	[K]	KILL		REW
	CHANGE		KLIST		RIGHT\$
	CHARACTERS\$	[L]	LEFT\$		RND
	CHR\$		LEN		ROPEN#
	CLOSE		LET		ROPEN/T
	CLOSE#		LIMIT		RUN
	CLOSE/T		LINE	[S]	SAVE
	CLR		LIST		SAVE/T
	CONSOLE		LIST/P		SET
	CONT		LN		SGN
	COPY/P		LOAD		SIN
	COS		LOAD/T		SIZE
	CSRH		LOCK		SPACE\$
	CSRV		LOG		SQR
	CURSOR	[M]	MID\$		STEP
[D]	DATA		MON		STOP
	DEF FN		MUSIC		STR\$
	DEF KEY	[N]	NEW		STRING\$
	DELETE		NEXT		SWAP
	DIM	[O]	ON	[T]	TAB
	DIR		OUT		TAN
	DIR/P	[P]	PAGE/P		TEMPO
[E]	END		PATTERN		THEN
	ERL		PEEK		TIS
	ERN		POINT		TO
	ERROR		POKE	[U]	UNLOCK
	EXP		POSH		USR
[F]	FAST		POSITION	[V]	VAL
	FOR		POSV		VERIFY
[G]	GET		PRINT	[W]	WOPEN#
	GOSUB		PRINT#		WOPEN/T
	GOTO		PRINT/P	[X]	XOPEN#
	GRAPH		PRINT/T		
[I]	IF	[R]	READ		

TABLE 1.1 All reserved words of the DISK BASIC interpreter SB-6510

Chapter 2

Instructions Unique to SB-6510

This chapter describes SB-6510 commands, statements and utilities which are not supported by the ordinary cassette BASIC interpreter SB-5510.

Command and statement format

Commands and statements must be coded according to the following conventions.

- *Small letters and reverse characters cannot be used for any commands and statements.*
- *Operands which must be specified by the programmer are indicated in italics.*
- *Items in brackets “\ \” may be omitted or repeated any number of times.*
- *Separators (commas, semicolons, etc.) must be correctly placed in the specified positions.*

2.1 Commands

2.1.1 DIR

Format : DIR <FD*d*>

d drive number : 1 through 4

Function : Displays the file directory of the diskette specified.

Description : When FD*d* is omitted, the value defaults to the number of the drive against which the last DIR FD*d* command was executed.

The contents of the directory are as follows:

- Volume number
For the master diskette, "MASTER" is displayed.
- The number of unused sectors remaining.
- Mode, lock condition and file name of each file on the diskette.

The four file modes are indicated with the following codes:

BTX : BASIC text file

BSD : BASIC sequential access file

BRD : BASIC random access file

OBJ : Object file

To indicate the lock condition, an asterisk is attached to the file mode.

Locked files cannot be overwritten or deleted, nor can their names be changed.

The file name specified during file creation must be always used to call the file.

When many files are contained on a diskette, the directory cannot be displayed in a single frame. The display is fixed once a frame is filled, and the cursor appears. The frame containing the remainder of the directory can then be brought to the screen by pressing the `↑` key. When the display is fixed, another command can be executed.

2.1.2 DIR/P

Format : DIR <FD*d*> / P

d drive number : 1 through 4

Function : Prints the directory of the diskette in drive *d* on the line printer.

2.1.3 SAVE

Format : SAVE <FDd@v,> “file name”

d drive number : 1 through 4

v diskette volume number

Function : Assigns the specified file name to the BASIC text contained in the text area and stores it on the diskette in the specified drive.

Description : The diskette on which the BASIC text is to be saved is specified with the FDd@v operand.

When this operand is omitted, the text will be stored on the diskette in the default drive.

“file name” consists of a string of up to 16 characters enclosed with quotation marks.

Example : SAVE “D” . . . Assigns the file name “D” to the BASIC text in the text area and stores it on the active diskette. The text is stored in the BTX file mode.

2.1.4 SAVE/T

Format : SAVE/T <file name >

Function : Assigns the file name to the BASIC text in the text area and stores it on the cassette tape.

2.1.5 VERIFY

Format : VERIFY <file name >

Function : This command automatically compares the program contained in the BASIC text area with its equivalent text (file name: “file name”) in the cassette tape file.

2.1.6 LOAD

- Format* : LOAD <FDD@v,> “file name”
d drive number : 1 through 4
v diskette volume number
- Function* : Loads the specified BASIC text file into memory from the specified diskette.
- Description* : The diskette is specified with the FDD@v operand.
When it is omitted, text is stored on the diskette in the default drive.
- Example* : LOAD FD2, “A” . . . Loads the BASIC text assigned the file name “A” from the diskette in drive 2 into the text area.
LOAD “TEXT 1” . . . Loads BASIC text “TEXT 1” from the diskette in the active drive into the text area.

2.1.7 LOAD/T

- Format* : LOAD/T <file name >
- Function* : Loads the BASIC text assigned the file name from the cassette tape into the text area.

2.1.8 RUN

- Format* : RUN <FDD@v,> “file name”
d drive number : 1 through 4
v diskette volume number
“file name” BTX file or OBJ file
- Function* : Loads the BASIC text (BTX) assigned the file name “file name” from the diskette, and then executes it from its beginning.
Therefore,
RUN “file name” = LOAD “file name” + RUN
Loads the machine language program (OBJ) assigned the file name “file name” from the diskette, and then executes the program at the start address. In such cases, system control is transferred from the BASIC interpreter to the machine language program.

2.2 File control statements

2.2.1 LOCK

Format : LOCK <FDd@v,> “file name”

d drive number : 1 through 4

v diskette volume number

Function : This statement locks a specified file.

Description : When a file is locked, requests to modify it will be denied. For example, the command prohibits DELETE or RENAME operations or writing of data in the case of random access files. It is good practice to lock files of a permanent or semi-permanent nature. The file mode symbols in the file directory display are followed by an asterisk to indicate protected files. (The write protect seal served as a hardware lock for an entire diskette)

2.2.2 UNLOCK

Format : UNLOCK <FDd@v,> “file name”

d drive number : 1 through 4

v diskette volume number

Function : This statement unlocks a specified file.

2.2.3 RENAME

Format : RENAME <FDd@v,> “file name 1”, “file name 2”

d drive number : 1 through 4

v diskette volume number

Function : This statement renames a specified file.

Description : To rename a file, its current name and its new name must be specified in this order. If a renamed file is identical in name and mode to any file currently stored on the same diskette, an error occurs.

The RENAME statement is prohibited for any locked file.

2.2.4 DELETE

Format : DELETE <FDd@v,> “file name”
d drive number : 1 through 4
v diskette volume number

Function : This statement deletes a specified file from the diskette.

Description : This statement is prohibited for any locked file. If you want to delete locked files, it is necessary to execute the UNLOCK statement first, then the DELETE statement.

2.2.5 CHAIN

Format : CHAIN <FDd@v,> “file name”
d drive number : 1 through 4
v diskette volume number

Function : This statement chains the program execution to BASIC text on the diskette.

Description : CHAIN FD2@7, “TEXT B” . . . Chains the program in the BASIC text area to BASIC program “TEXT B” on the diskette volume 7 in drive 2. That is, program “TEXT B” is loaded in the BASIC text area and program execution is started at its beginning. Before the text is loaded, the BASIC text area is cleared but all variable values and contents of user functions are given to the program. The CHAIN statement has the same function as GOTO “file name”.
CHAIN “PROGRAM 3” . . . Chains the program in the BASIC text area to program “PROGRAM 3” on the diskette in the active drive.

2.2.6 SWAP

- Format* : SWAP <FDd@v,> “file name”
d drive number : 1through 4
v diskette volume number
- Function* : This statement swaps the program execution to BASIC text on the diskette.
- Description* : SWAP FD2@7, “TEXT S-R” . . . Swaps the current program for BASIC program “TEXT S-R” on diskette volume 7 in drive 2. The current program text is saved on the diskette in the drive specified in the last DIR FDd command, then program “TEXT S-R” is loaded into the text area and is executed from its beginning. When the swapped program is finished, the saved program is loaded again and program execution is started at the statement following the SWAP statement. The values of variables and the contents of user functions are transferred between the two program. No SWAP statement can be used in a swapped program. The SWAP statement has the same function as GOSUB “file name”.

■ BSD (BASIC Sequential access Data file) control

2.2.7 WOPEN#

- Format* : WOPEN #l, <FDd@v,> “file name”
l logical number
d drive number : 1 through 4
v diskette volume number
- Function* : This statement opens a diskette file to allow a sequential access file to be written on the diskette.
- Description* : WOPEN #3, FD2@7, “SEQ DATA 1” . . . Defines the file name of a BSD (BASIC sequential access data file) to be created as “SEQ DATA 1” and opens it with logical number 3 assigned on diskette volume 7 in drive 2.

2.2.8 PRINT#

Format : PRINT # l , d_1 < , d_2 , . . . , d_v >
 l logical number
 d_i write data

Function : This statement writes the data d_1 , d_2 . . . d_v (numeric data or string data) in order in the BSD assigned logical number l which was opened by a WOPEN# statement.

Description : PRINT #3, A, A\$. . . Writes the contents of variable A and string variable A\$ in order in the BSD assigned logical number 3 which was opened by a WOPEN# statement.

2.2.9 CLOSE#

Format : CLOSE <# l >
 l logical number

Function : This statement closes a BSD assigned logical number l .

Description : CLOSE #3 . . . Closes the BSD assigned logical number 3 which was opened by the WOPEN #3 statement.

By closing the BSD, the BSD which has the file name defined in the WOPEN# statement is created on the specified diskette, and the logical number assigned is made undefined.

2.2.10 KILL#

Format : KILL <# l >
 l logical number

Function : This statement kills a BSD assigned logical number l .

Description : KILL #3 . . . kills the BSD assigned logical number 3 by the WOPEN# statement. Logical number 3 is made undefined.

2.2.11 ROPEN#

- Format* : ROPEN #*l*, <FD*d*@*v*,> “file name”
l . . . logical number
d . . . drive number : 1 through 4
v . . . diskette volume number
- Function* : This statement opens a diskette file to allow a sequential access file to be read from the diskette.
- Description* : ROPEN #4, FD2@7, “SEQ DATA 1” . . . Opens BSD “SEQ DATA 1” on diskette volume 7 in drive 2 with logical number 3 assigned to read data in BSD.

2.2.12 INPUT#

- Format* : INPUT #*l*, *v*₁ < , *v*₂, . . . , *v*_{*n*} >
l . . . logical number
*v*_{*i*} . . . read data
- Function* : This statement reads data stored in the specified BSD in order and assigns to variables *v*₁, *v*₂ . . . *v*_{*n*} (or array elements).
- Description* : INPUT #4, A(1), B\$. . . Reads data sequentially from the beginning of the BSD assigned logical number 4 which was opened by the ROPEN# statement and substitutes numerical data into array element A(1) and string data into string variable B\$.
 CLOSE #4 statement closes the BSD assigned logical number 4 and the logical number undefined.

■ BRD (BASIC Random access Data file) control**2.2.13 XOPEN#**

Format : XOPEN # l , <FD d @ v ,> “file name”

l . . . logical number

d . . . drive number : 1 through 4

v . . . diskette volume number

Function : Generally, XOPEN# opens a BRD for writing and reading data (CROSS open).

Description : XOPEN #5, FD3@18, “DATA R1” . . . This statement cross-opens BRD “DATA R1” on diskette volume 18 in drive 3 with logical number 5 assigned or, if the file does not exist on the diskette, cross-opens a BRD by defining its file name as “DATA R1” to create it on the diskette with logical number 5 assigned.

2.2.14 PRINT# ()

Format : PRINT # l (n), d_1 <, d_2 , . . . , d_n >

l . . . logical number

n . . . item expression

d_i . . . write data

Function : This statement writes numeric or string data on elements n , $n + 1$, . . . , $n + n$ of the BRD assigned logical number 1 which was opened by the XOPEN# statement.

Description : PRINT #5(11), R(11) . . . Writes the contents of 1-dimensional array element R(11) on element 11 of the BRD assigned logical number 5 which was opened by the XOPEN# statement.

PRINT #5(20), AR\$, ASS\$. . . Writes the contents of string variables AR\$ and ASS\$ on element 20 and element 21 of the BRD assigned logical number 5, respectively. All BRD elements have a fixed length of 32 bytes and, if the length of string variable exceeds 32 bytes, the excess part is discarded.

2.2.15 INPUT#()

Format : INPUT # $l(n)$, v_1 $\langle, v_2, \dots, v_n \rangle$

l . . . logical number

n . . . item expression

v_i . . . read data

Function : This statement reads data stored in the specified elements of the specified BRD.

Description : INPUT #5(21), R\$. . . Reads the content of element 21 of the BRD assigned logical number 5 which was opened by the XOPEN# statement into string variable R\$.

INPUT #5(11), A(11), A\$(12) . . . Reads the contents of element 11 and element 12 of the BRD assigned logical number 5 into linear numeric array element A(11) and linear string array element A\$(12), respectively.

CLOSE #5 statement closes the BRD assigned logical number 5 which was opened by the corresponding XOPEN# statement.

KILL #5 statement kills the BRD assigned logical number 5 and the logical number undefined.

CLOSE . . . Closes all open files.

KILL . . . Kills all open files.

2.2.16 IF EOF(#) THEN

Format : IF EOF(# l) THEN lr (or *statement*)

l . . . logical number

Function : Transfers program control to the routine starting to specified line number lr if an EOF (End of file) is detected when as INPUT# statement is executed against a BSD or a BRD.

Example : IF EOF(#5) THEN 1200

2.3 Error processing control

2.3.1 ON ERROR GOTO

- Format* : ON ERROR GOTO *lr*
lr reference line number : error processing routine
- Function* : This statement declares the number of the line to which program execution is to be moved in order to correct errors.
- Description* : Declaring an error processing routine with the ON ERROR GOTO statement allows errors to be corrected during program execution without the system returning to the BASIC command level. When the ON ERROR GOTO statement is executed, program execution will be moved to <error processing routine> if any error has occurred. This enables the error number (ERN) and the number of the line on which the error occurred (ERL) to be ascertained, and allows subsequent processing to be performed in accordance with the IF ERN or ERL statements. The RESUME statement serves to move program execution back to the point at which the error occurred.
Execution of a new ON ERROR GOTO statement invalidates any preceding one.

2.3.2 IF ERN

- Format* : IF ERN *expression* THEN *lr*
IF ERN *expression* THEN *statement*
IF ERN *expression* GOTO *lr*
lr reference line number
- Function* : This statement ascertains the identification numbers of errors, and causes branching when those numbers are ones specified.
- Description* : When an error occurs, the corresponding error number is placed in system variable ERN (see the Error Table on page 88). This enables an IF ERN statement in an error correcting routine declared by the ON ERROR GOTO statement to determine what type of error has occurred. The IF ERN statement may be used in either of the following forms:
(1) IF <relational expression of ERN> GOTO, or
(2) IF <relational expression of ERN> THEN *statement* or *lr*.
(See the descriptions of the IF ~ THEN and IF ~ GOTO statements.)

Example : The statement shown below causes program execution to jump to line 1200 when Error 5 (String Overflow) occurs, indicating that the string length exceeded 255 characters.

```
800 IF ERN=5 THEN 1200
```

2.3.3 IF ERL

Format : IF ERL *expression* THEN *lr*
IF ERL *expression* THEN *statement*
IF ERL *expression* GOTO *lr*
lr *reference line number*

Function : This statement determines the number of the line on which an error has occurred and causes branching to a specified line.

Description : Since system variable ERL is loaded with the number of the line on which an error occurred, the IF ERL statement in the routine declared by the ON ERROR GOTO statement is able to ascertain this line number from system variable ERL. The IF ERL statement, like the IF ERN statement, may be used in two forms: IF ~ THEN or IF ~ GOTO.

Example : The statement shown below causes program execution to jump to line 1300 when an error occurs on line 250.

```
810 IF ERL = 250 THEN 1300
```

2.3.4 RESUME

Format : RESUME <NEXT>
RESUME *lr*
lr reference line number or 0

Function : This statement returns program execution to the main program after correction of an error.

Description : The system holds the number of the line on which the error occurred in memory and returns program execution to that line or to another specified line after the error is corrected.

The RESUME statement may be used in any of the following four forms:

RESUME: This returns program execution to the statement in which the error occurred.

RESUME NEXT: This returns program execution to the statement just after the one in which the error occurred.

RESUME <line number>: This returns program execution to the line specified by <line number>.

RESUME 0: This returns program execution to the beginning of the program, or to the line with the smallest line number.

If the system encounters any RESUME statement when there is no error condition, Error 21 (RESUME - no ERROR) will occur.

2.4 Use of utility programs

Utility programs "Filing CMT" (OBJ) and "Utility" (OBJ) are stored on the master diskette together with DISK BASIC interpreter SB-6510, MONITOR SB-1510 and some application programs, as shown in Figure 2.1 the file directory of the master diskette.

```

DIR FD1
MASTER 702 SECTOR FREE
OBJ* "Filing CMT"
OBJ* "Utility"
BTX* "AUTO RUN"
BTX* "MUSIC"
BTX* "Sporhythm"
BTX* "Queen"
BTX* "Tri Function"
BTX* "Graphic pattern"
BTX* "Die Fraktur"
BTX* "3D-PL0T"
BTX* "Data Processing"
BTX* "Explanation-D.P"
BTX* "String-Machine"
BTX* "Variable Data"
BSD "DATA-DEC2"
Ready

```

FIGURE 2.1

In the following paragraphs, use of utility programs are explained.

2.4.1 Use of utility program "Filing CMT"

This utility program transfers machine language program from cassette file to the diskette as it is. To call this utility program, enter

```
RUN "Filing CMT"
```

The display screen is as shown in Figure 2.2.

```

* TRANSFER FROM CMT (OBJECT TAPE) TO FD *
SET TAPE! OK?
(B KEY : BOOT START)
DRIVE NO. ☒

```

FIGURE 2.2

Set ready the cassette tape file which has to be transferred into the diskette, and specify the drive number.

The following example transfers BASIC interpreter SB-5510 from the cassette file to the diskette in drive 2.

```
* TRANSFER FROM CMT (OBJECT TAPE) TO FD *
  SET TAPE! OK?
    (B KEY : BOOT START)
  DRIVE NO. 2
  LOADING BASIC SB-5510
    (R) KEY : RESTART
    OTHER KEY : BOOT START
```

FIGURE 2.3

You obtain the object file (OBJ) "BASIC SB-5510" on the diskette in drive 2. Therefore, to call BASIC interpreter SB-5510 from the diskette file, simply enter

```
RUN "BASIC SB-5510"
```

2.4.2 Use of utility program "Utility"

This utility program has two functions, that is, initializing diskettes and copying diskettes. To call this utility program, enter

```
RUN "Utility"
```

The display screen is as shown below.


```
** UTILITY **
[COMMAND TABLE]
DISKETTE INIT   : I
SLAVE-DISK INIT : S
DISKETTE COPY  : C
BOOT START     : B
? 
```

FIGURE 2.4

When a I command is entered, the display is as shown in Figure 2.5.

```
[ COMMAND TABLE ]
DISKETTE INIT   : I
SLAVE-DISK INIT : S
DISKETTE COPY   : C
BOOT START      : B
? I
DRIVE NO. ❷
```

FIGURE 2.5

The utility program requests the operator to specify drive number.

When a new diskette is used, it must first be initialized (that is, formats the diskette so that data are able to be written or read). During initialization, the diskette is formatted.

S command assigns a volume number to a initialized diskette for making a slave diskette. Figure 2.6 shows an example where the slave diskette in drive 1 is made with volume number 25 assigned.

```
[ COMMAND TABLE ]
DISKETTE INIT   : I
SLAVE-DISK INIT : S
DISKETTE COPY   : C
BOOT START      : B
? S
VOLUME NO. 25
```

FIGURE 2.6

Any number from 1 through 127 can be specified for the volume number. Different numbers must be assigned to each diskette so that a diskette can be specified with its volume number in a logical open statement.

C command copies a diskette. The following example copies the diskette in drive 1 on diskette in drive 2.

```
[ COMMAND TABLE ]
DISKETTE INIT   : I
SLAVE-DISK INIT : S
DISKETTE COPY   : C
BOOT START      : B
? C
FROM
DRIVE NO. 1
TO
DRIVE NO. 2
```

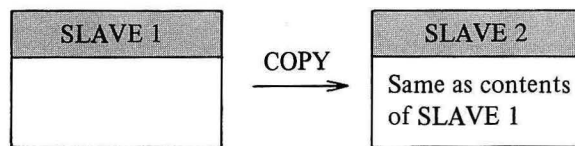


FIGURE 2.7 Slave diskette copying

Any number of submaster diskettes can be made by copying the master diskette using this diskette copy command. However, submaster diskettes cannot be made by copying another submaster diskette.

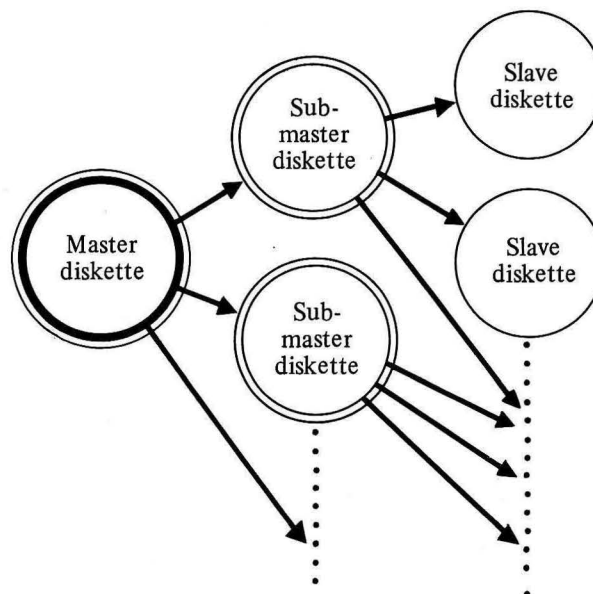


FIGURE 2.8

NOTE: It is recommended that a write protect seal be placed on the original diskette to prevent it from being accidentally arased.

ERROR INDICATIONS : DISK ERROR = 50 The disk drive is not ready.
DISK ERROR = 41 Disk drive hardware error. .

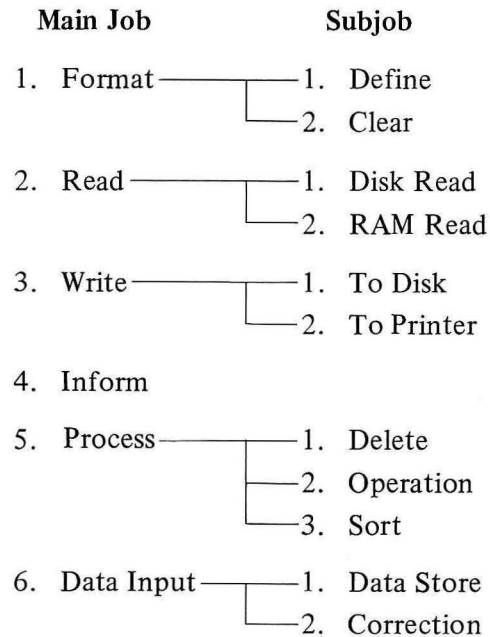
Chapter 3

Data Processing Application

This chapter introduces a typical example of file control program using DISK BASIC in order that the user may utilize the fundamental programming technique described in the example.

Data processing application

This program performs the following: data file generation, file-based calculations, data sorts and data list generation. This program is designed so that it is applicable to a variety of situations, such as inventory control and generation of a list of student records. The user determines the application. This program consists of the following jobs:



Data file format

This program generates a data file in the format shown in the Figure 3.1. The maximum number of fields N is 10 and the maximum number of data records M is 150. A group of data records which are arranged on the same line as in this Figure is represented by a data record number. For example, data record 2 represents the group of data fields on the highlighted line in the Figure 3.1. Data is input as a unit consisting of the group of data fields represented by the data record number. Deletion of a data record number means that the group of data fields represented by the data record number is deleted.

	Field 1	Field 2			Field N
1	Data	Data			Data
2	Data	Data			Data
3	Data	Data			Data
⋮	⋮	⋮			⋮
⋮	⋮	⋮			⋮
M-1					
M	Data	Data			Data

FIGURE 3.1

Data length

The length of the input data must be specified for every field. A length equal to or greater than that of the longest data element belonging to a field must be specified for the field.

Data attribute

Whether input data is numeric data or alphanumeric data must be specified. The program processes alphanumeric data as string data. Only the following codes are significant.

Numeric : 01H~04H, 07H, 08H, 0DH, 1BH, 20H, 2BH, 2DH, 2EH, 30H~39H (according to ASCII)

Alphanumeric : 01H~04H, 07H, 08H, 0DH, 1BH, 20H~7EH (according to ASCII)

Program Execution

a) Setting date

Enter the date from the keyboard in the order month, day and year. The length of the month and day is up to 2 digits and that of the year is up to 4 digits. For example, to enter Nov. 9, 1981, press keys as follows.

1 1 ENT → 9 ENT → 1 9 8 1 ENT

or

1 1 ENT → 9 ENT → 8 1 ENT

The CR key may be pressed instead of the ENT key. See Figure 3.2.

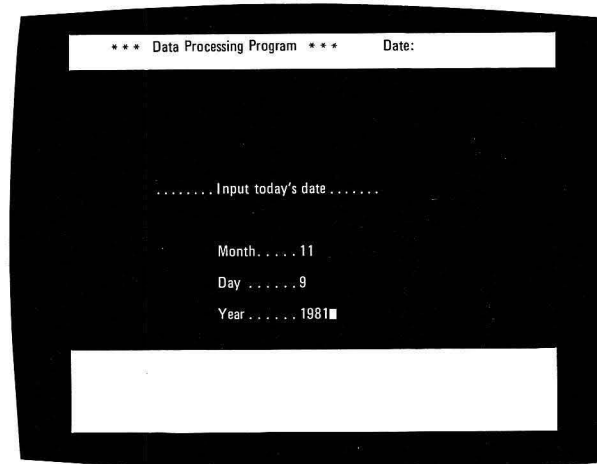


FIGURE 3.2

b) Job selection (job selection routine)

The character string "MAIN JOB" flashes on the bottom line of the CRT display screen. Select from among the 6 main jobs by pressing one of keys to . See Figure 3.3.

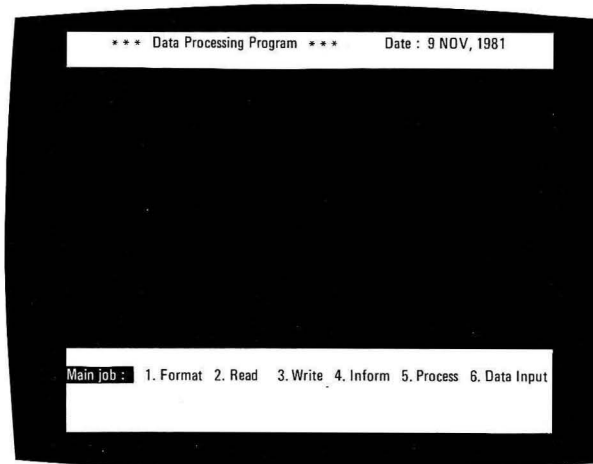


FIGURE 3.3

After the main job is selected, select a subjob. See Figure 3.4.

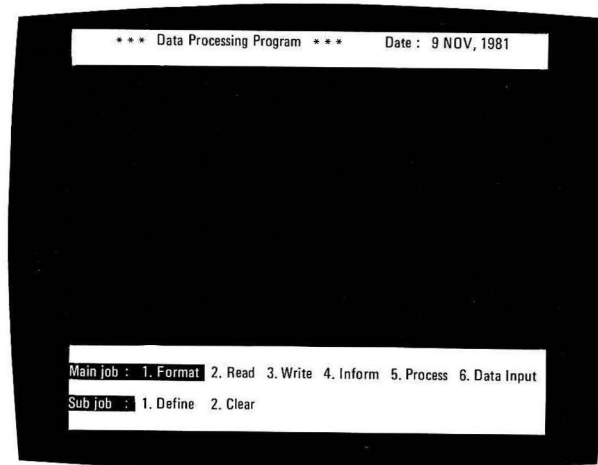


FIGURE 3.4

Notes: This program and data generated by this program must be stored in the same diskette. During execution of this program, the or key must be pressed to conclude data entry from the keyboard to make the key input data significant.

Job 1: Format

(1) Define

Defines the name, length and attribute for every field according to data entered from the keyboard. Data entered must satisfy the following conditions:

Field name -- Up to 16 alphanumeric characters

Data length -- Up to numerics 79. The total length of all fields must be 79 or less.

Data attribute -- "1" for numeric data and "0" for string data

The above items must be specified for every field required. See Figure 3.5. Pressing the **TAB** key returns control to the job selecting routine. See Figure 3.3.

```

*** Data Processing Program ***      Date : 9 NOV, 1981
Information : Input Field Name      (Up to 16 characters)

Field : 1      Field Name = Student Number
              Field Length = 4
              Field Attribute = 0

Field : 2      Field Name = Student Name
              Field Attribute = 1

Main job : 1. Format 2. Read 3. Write 4. Inform 5. Process 6. Data Input
Sub job  : 1. Define 2. Clear
  
```

FIGURE 3.5

(2) Clear

Clears the file and cancels all field definitions. This subjob is to be executed before defining a new data file.

Job 2: Read

(1) Disk read

Displays the names of the first 6 data files stored on the disk. The operator must enter a number corresponding to the file to be read when it is displayed on the screen. If the name of the file to be read is not found among the 6 names displayed, press the space bar and the program displays the names of the next 6 files stored on the disk. Repeat until the desired file name is found. When no more files remain on the disk, the program displays "END" at the lower right of the last file name. When any files remain, it displays "TO BE CONTINUED" at the lower right of the last file name. See Figure 3.6.

*** Data Processing Program ***					Date : 9 NOV, 1981
[No.]	[Field Name]	[Date]	[No. of Data]	[No. of Field]	
1.	Marks List (25)	9 NOV, 1981	25	8	
2.	Marks List (15)	9 NOV, 1981	15	8	
3.	Average . . . (15)	9 NOV, 1981	15	9	
4.	Average 15	9 NOV, 1981	15	9	
5.	Average Order . . . 15	9 NOV, 1981	15	9	
. End					
Main job : 1. Format 2. Read 3. Write 4. Inform 5. Process 6. Data Input					
Sub job : 1. Disk Read 2. RAM Read					
Select : Specify one of above files [1 to 5]					

FIGURE 3.6

When a file name is selected, the program reads and stores the corresponding data file into memory. Pressing the **TAB** key returns control to the job selection routine.

(2) RAM read

Displays data in the file in the memory.

This subjob operates in either the automatic read or the step read mode. Enter **1** or **2** from the keyboard to select the mode.

In the automatic read mode, all data records in the data file are displayed consecutively. By pressing the space bar during execution in the automatic read mode, display operation can be suspended. Pressing the space bar again restarts display operation. See Figure 3.7.

*** Data Processing Program ***					Date : 9 NOV, 1981				
No. 1	006	Kristy McNichol	5	98	85	82	75	85	85
No. 2	004	Dustin Hoffman	5	78	84	65	75	94	79.2
No. 3	011	Jane Fonda	5	65	76	85	95	74	79
No. 4	014	Clint Eastwood	5	76	85	64	68	85	75.6
No. 5	010	Faye Dunaway	5	45	76	84	72	85	72.4
Main job : 1. Format 2. Read 3. Write 4. Inform 5. Process 6. Data Input									
Sub job : 1. Disk Read 2. RAM Read									
Select : 1. Auto Read 2. Step Read Stop									

FIGURE 3.7

In the step read mode, data records in the data file are displayed one by one. Pressing the space key displays the next data record. See Figure 3.8.

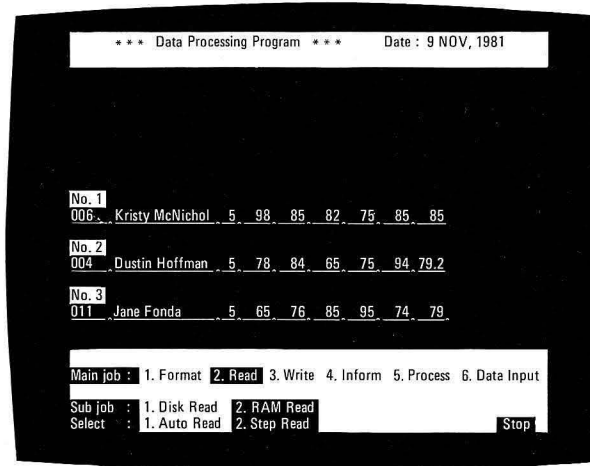


FIGURE 3.8

The mode can be alternated by pressing or while the operation is suspended. Pressing the key returns control to the job selection routine.

Job 3: Write

(1) To disk

Stores the data file on the disk. The operator must specify a file name of up to 16 characters. File names which have already been specified cannot be used. See Figure 3.9.

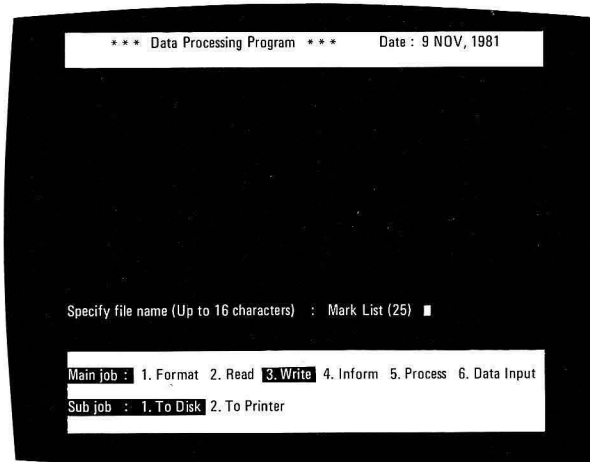


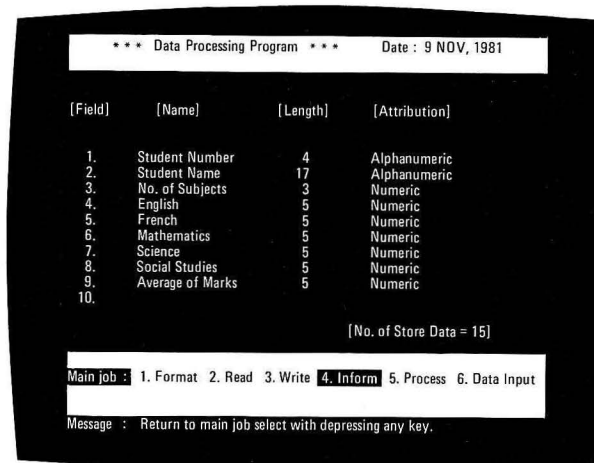
FIGURE 3.9

(2) To printer

Prints a data file listing. Pressing the **TAB** key during execution stops printing and returns control to the job selection routine.

Job 4: Inform

Displays field definitions defined by job 1 on the CRT screen. See Figure 3.10.



[Field]	[Name]	[Length]	[Attribution]
1.	Student Number	4	Alphanumeric
2.	Student Name	17	Alphanumeric
3.	No. of Subjects	3	Numeric
4.	English	5	Numeric
5.	French	5	Numeric
6.	Mathematics	5	Numeric
7.	Science	5	Numeric
8.	Social Studies	5	Numeric
9.	Average of Marks	5	Numeric
10.			

[No. of Store Data = 15]

Main job : 1. Format 2. Read 3. Write 4. Inform 5. Process 6. Data Input

Message : Return to main job select with depressing any key.

FIGURE 3.10

Pressing any key returns control to the job selection routine.

Job 5: Process

(1) Delete

Deletes a group of successive data records. The operator must specify the first record number of the group and the number of data records to be deleted. Data records following the group deleted are moved up to the data record preceding the group deleted. See Figure 3.11. Pressing the **TAB** key returns control to the job selection routine.

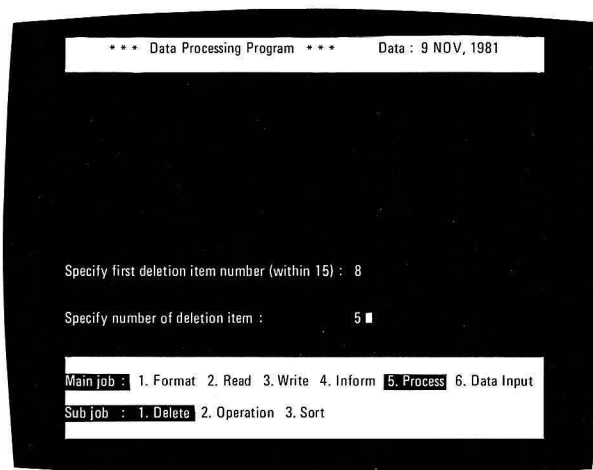


FIGURE 3.11

(2) Operate

Performs arithmetic operations concerned with two fields of the data file. The operator must specify the type of arithmetic operation to be performed. Keys **1** through **4** correspond to addition, subtraction, multiplication and division, respectively. The two fields to be subjected to operations and the field in which the results are to be stored must also be specified. For example, when each data element in field 1 is to be multiplied by the corresponding data element in field 3 and each result is to be stored in field 5, specify as shown in Figure 3.13. If field 5 is not defined, it must be defined by job 1 before this job is executed.

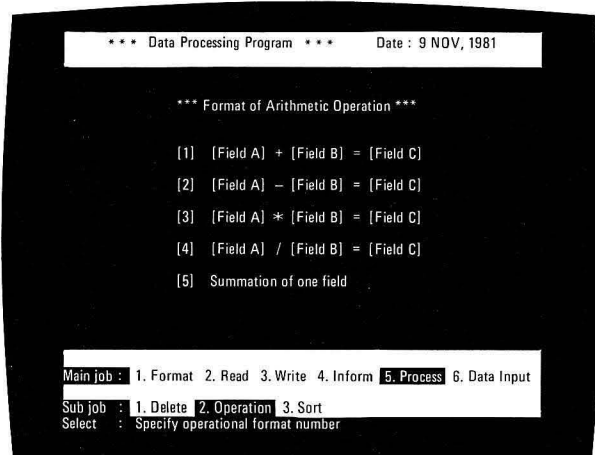


FIGURE 3.12

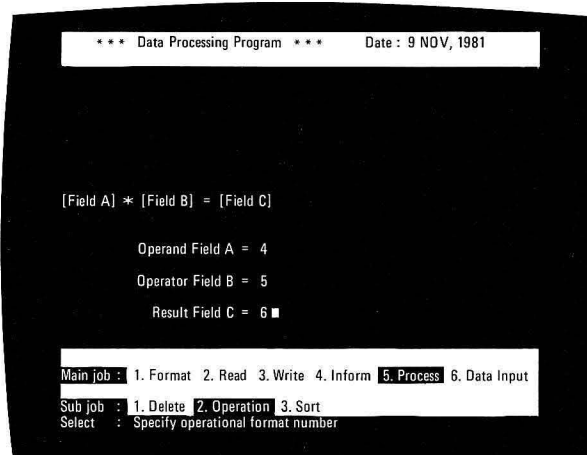


FIGURE 3.13

It is possible to store the result of addition of the data elements in field 1 and field 2 in field 1. In this case, the previous contents of field 1 are lost. An appropriate length must be defined for fields in which the results of arithmetic operations are to be stored. If the length of the result exceeds the defined length of the field, the result is insignificant. The program ignores the second decimal place and on in the result.

Pressing when the operation is selected causes the program to add all data elements in a field and obtain their average.

All fields subjected to arithmetic operations must be numeric fields.

Pressing the key returns control to the job selection routine.

(3) Sort

Rearranges the order of data in the specified numeric field. This subjob operates in either of two modes.

In the descent mode, it arranges data from the largest downward. In the ascend mode, it arranges data from the smallest upward. Keys or set the descent or ascend mode, respectively. See Figure 3.14.

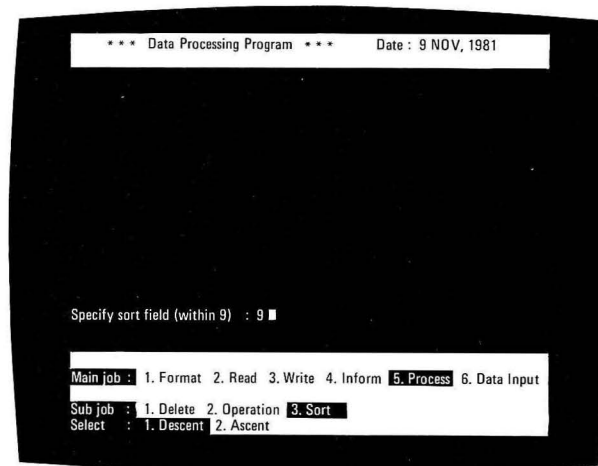


FIGURE 3.14

Pressing the key returns control to the job selection routine.

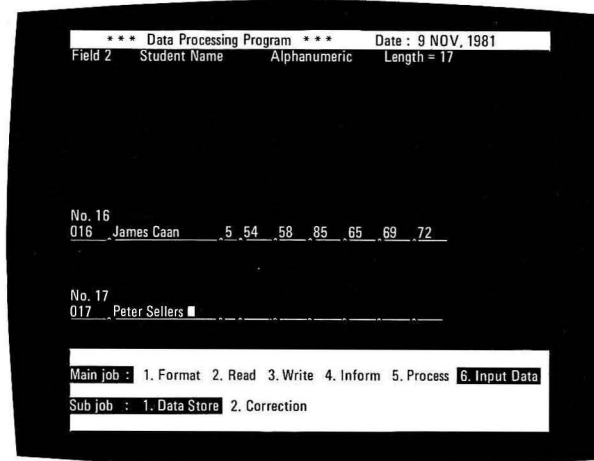


FIGURE 3.16

Pressing the **TAB** key returns control to the job selection routine.

(2) Correction

Corrects data which has already been input.

First specify the record number, then the program displays the specified data record. Then, press the **CR** or **ENT** key until the cursor is positioned in the field containing the data to be corrected. Enter the corrected data then advance the cursor to the next field to be corrected by pressing the **CR** or **ENT** key. For example, to correct the data in field 5, press the **CR** or **ENT** key four times and enter the corrected data. After the data has been entered, be sure to press the **CR** or **ENT** key to advance the cursor then press the **TAB** key. Corrections are made significant by pressing the **TAB** key. See Figure 3.17.

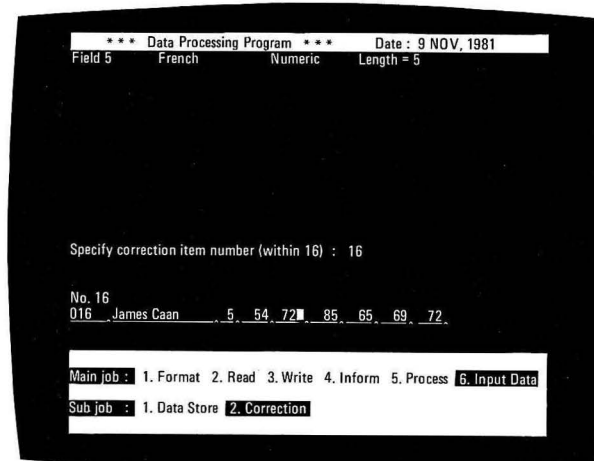


FIGURE 3.17

Pressing the **TAB** key returns control to the job selection routine.

Primary Program Variables

AA : Number of data files already stored on the disk.
 TD\$: Date
 JM : Main job number
 SJ : Subjob number
 NF : Number of fields defined ($N1 = NF - 1$)
 P1 : Number of data records ($P1 = P - 1$)
 A\$(J,I): Data stored in data record I, field (J + 1)
 A\$(J, 0) stores the name of field (J + 1).
 B(I) : The length of the data field (I + 1)
 A(I) : The attribute of the data field (I + 1)
 U : Sum of the length of defined data fields.

Machine Language Routine

Machine language data are stored in the memory area from \$FE7E to \$FFFF. The area contains three subroutines.

(1) Subroutine starting at \$FE7E (keyboard input routine)

This subroutine obtains data entered from the keyboard and stores it in the area corresponding to string variable IP\$. To call this subroutine, the following data must be set in advance.

Addresses \$FFFE

and \$FFFF : Video-RAM address (\$D550)

Address \$FFFC : Starting address of the area in which data is to be stored, written in offset from \$D550.

Address \$FFFA : Input data attribute
 1 = numeric data
 0 = alphanumeric data

Address \$FFF B : Input data length

Line numbers 7000 to 7050 are assigned to this subroutine. This subroutine is invoked by GOSUB 7500 or GOSUB 7600.

(2) Subroutine starting at \$FFDD

This subroutine shifts data on the 3rd through 20th line on the CRT display screen upward one line when the display mode is in the 80 characters/line mode. As a result, the 20th line becomes a space line. This subroutine is invoked by GOSUB 7070 or GOSUB 7080.

(3) Subroutine starting at \$FFEA

This subroutine obtains one character entered from the keyboard and stores its corresponding ASCII code in \$FFF9.

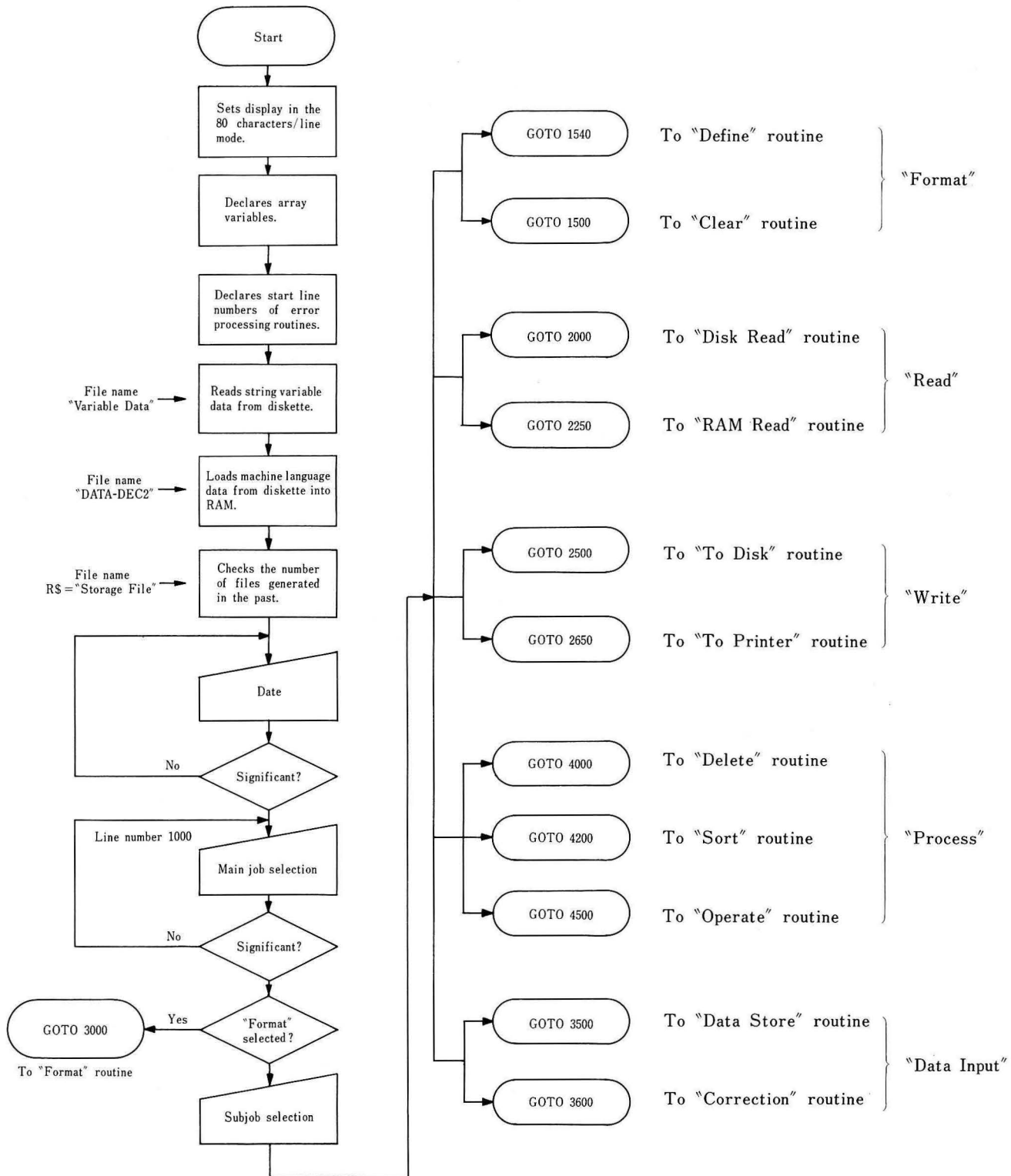
Line number 7780 is assigned as the start address of this subroutine. It is called by GOSUB 7780. This subroutine is also invoked by GOSUB 7070 or GOSUB 7800.

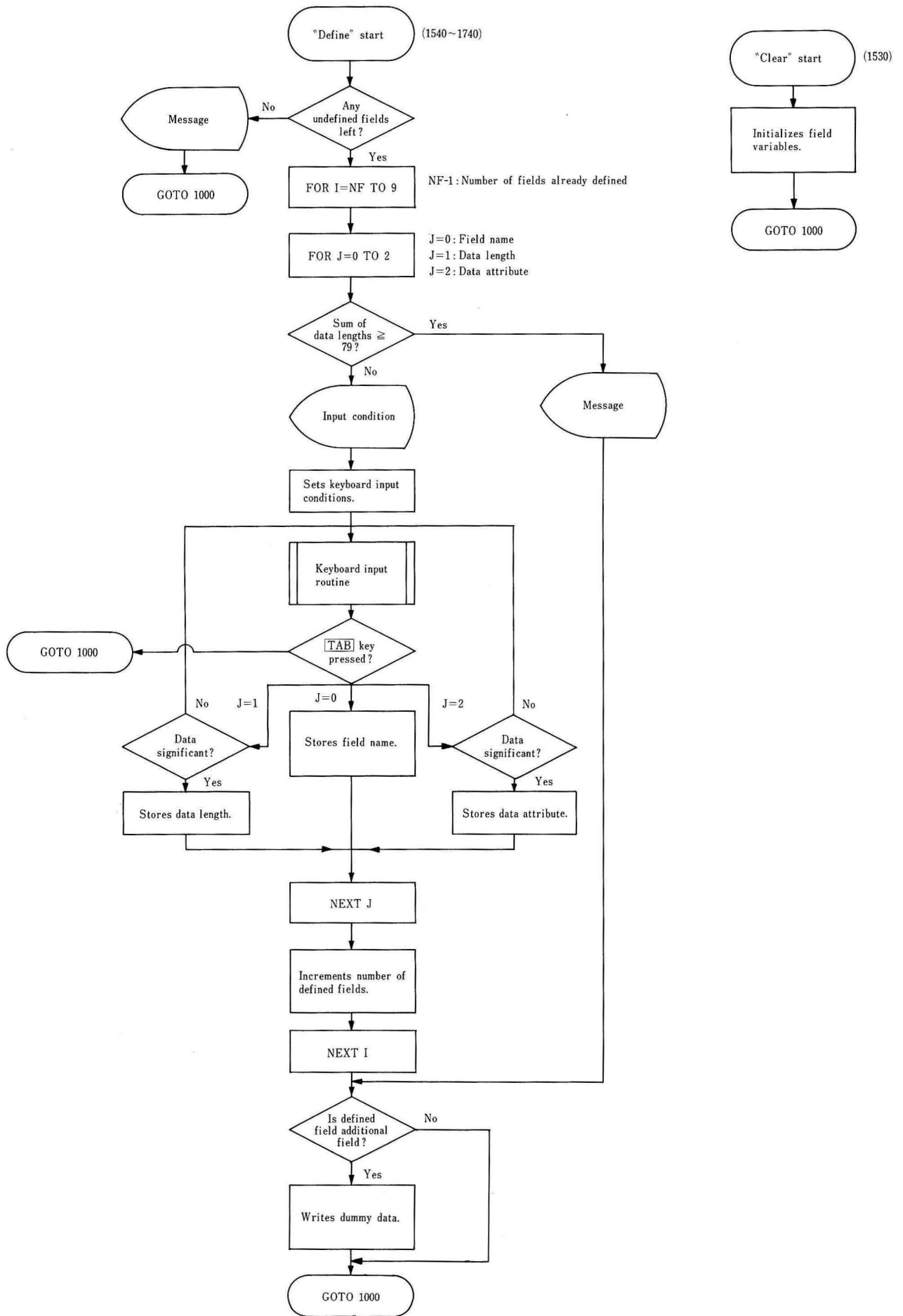
Structure of Files Generated

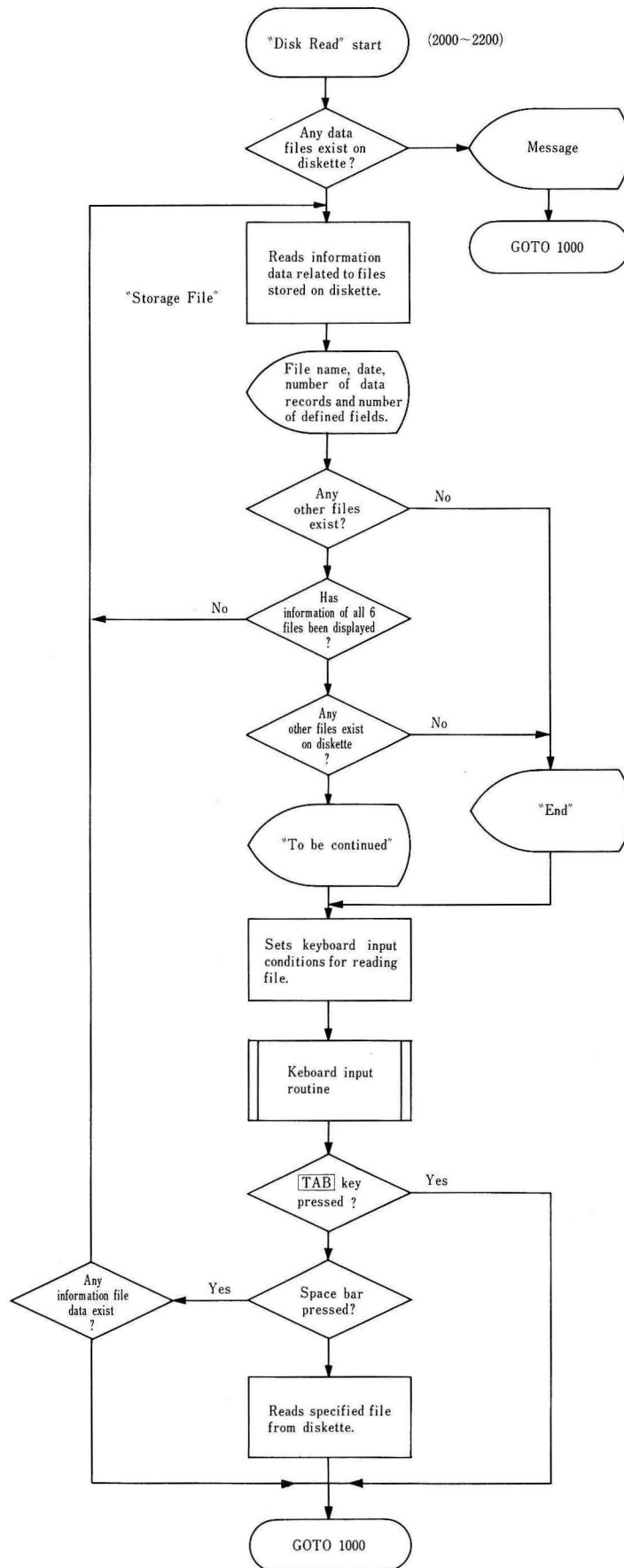
Each data file generated is stored as a serial data file (BSD) on the diskette with an appropriate name assigned. At the same time, a random data file (BRD) containing information related to each data file, i.e., the file name, date, number of fields defined and number of data records is generated and stored with the file name "Storage File" assigned.

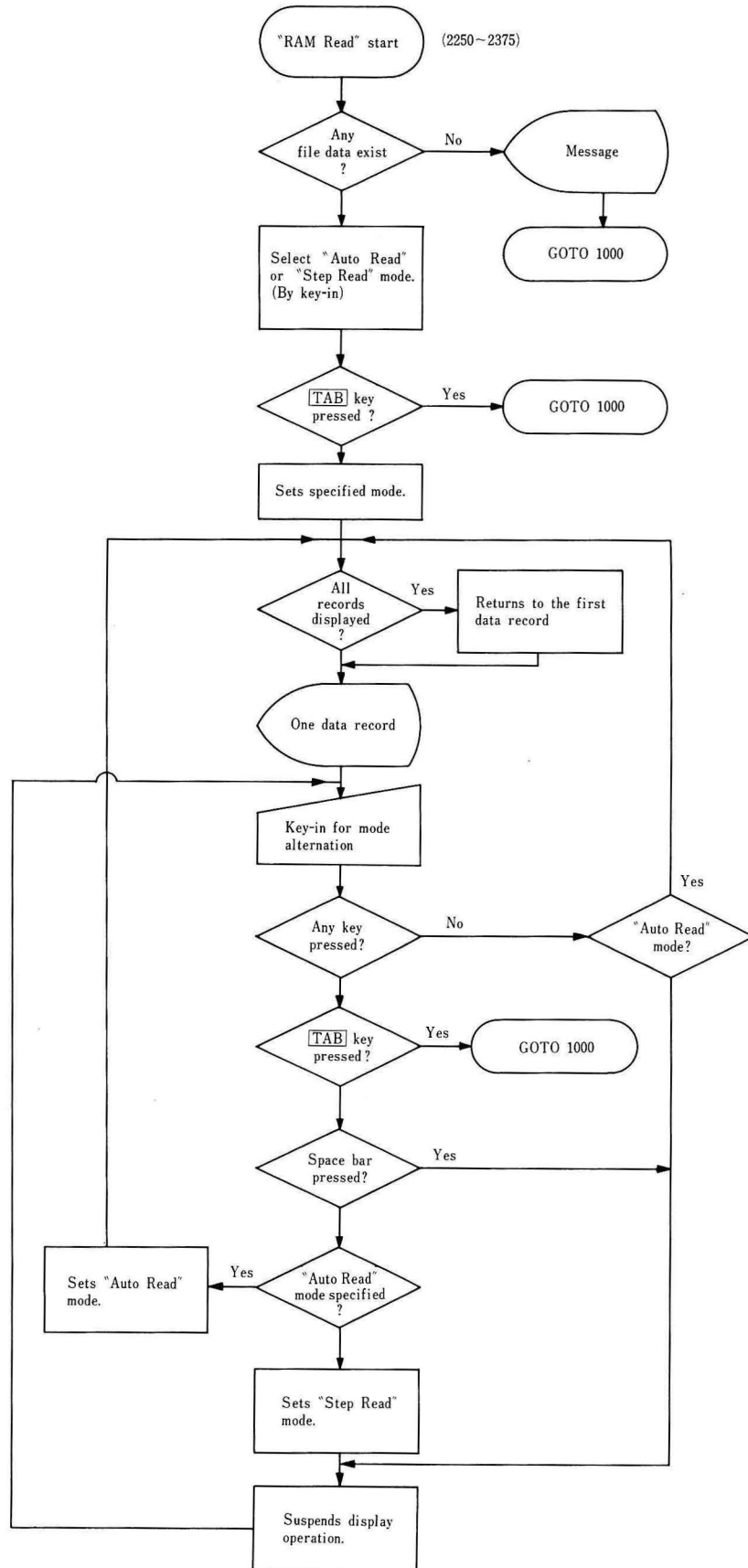
Flowchart

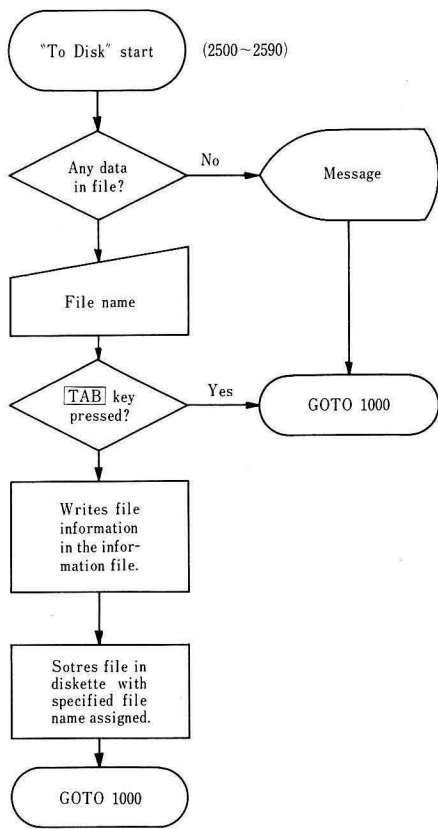
The flowchart of this program is shown in the following. The program listing is shown later. The following flow chart is for reference only and does not include all details of the program.



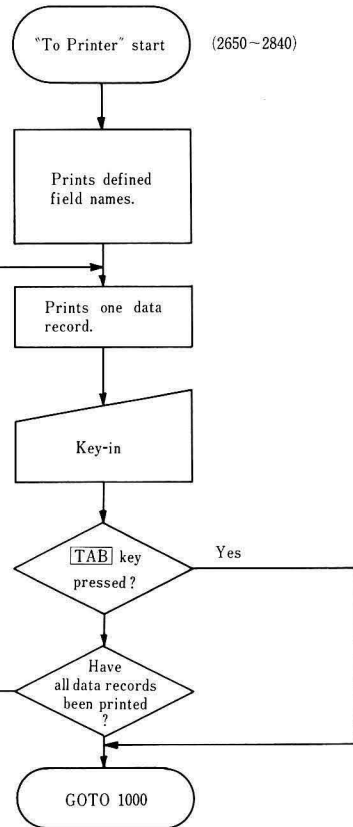
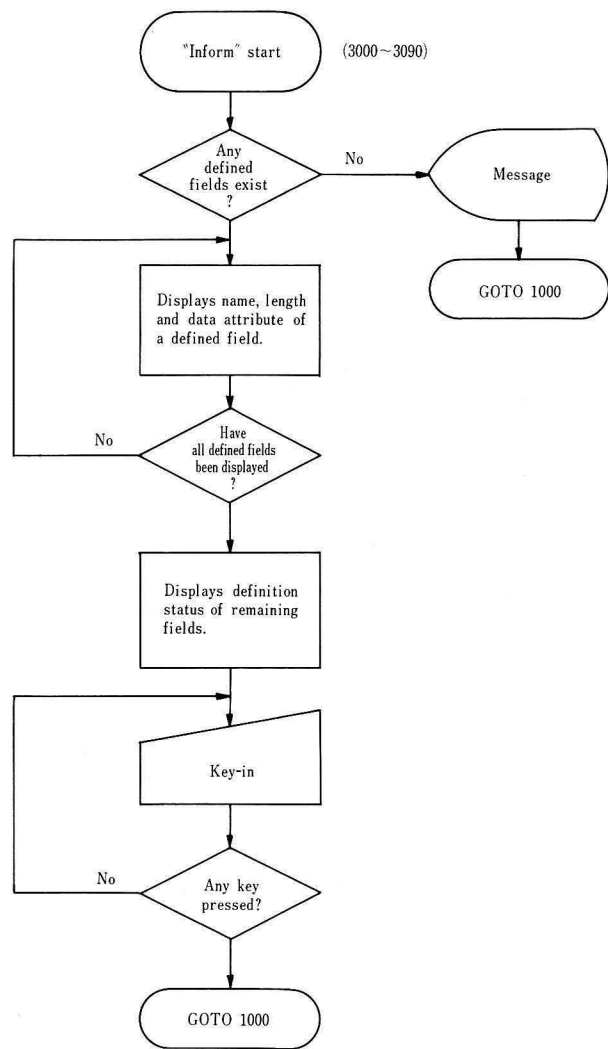


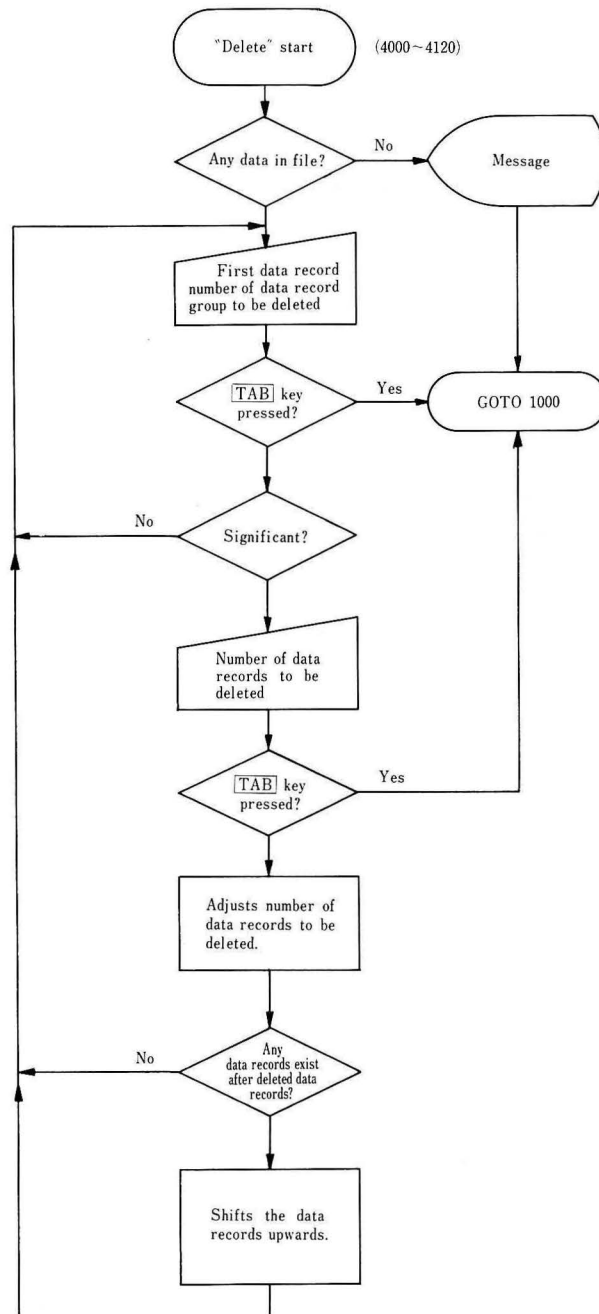


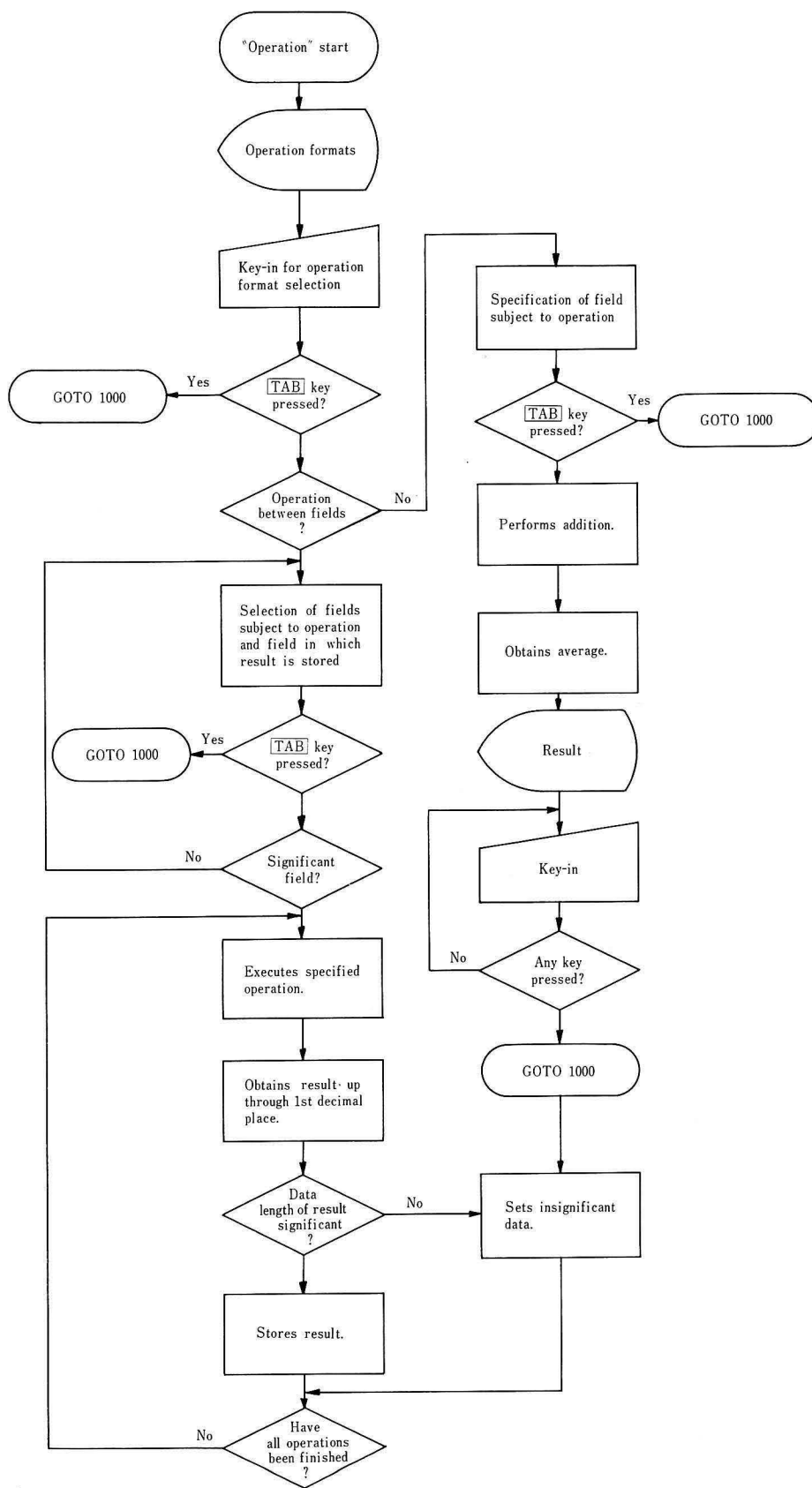


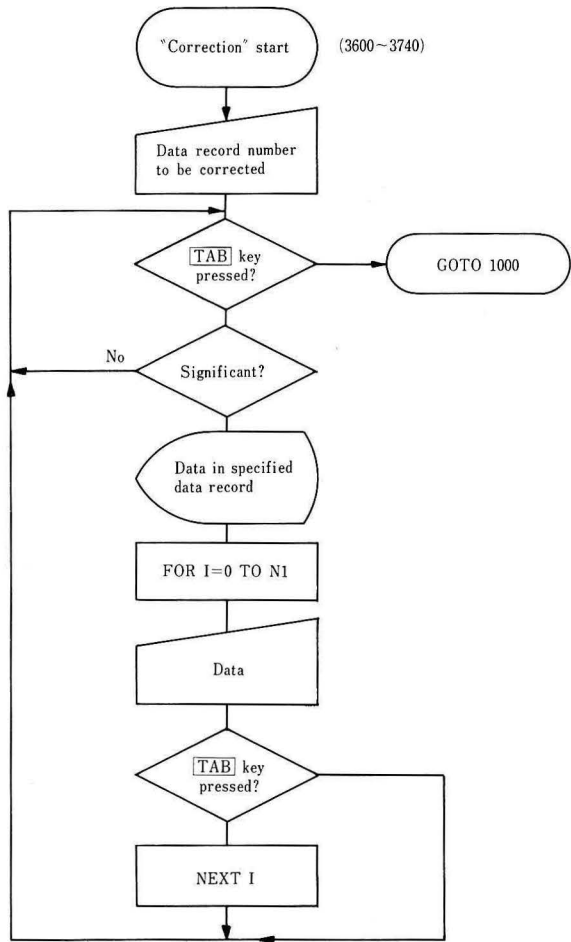
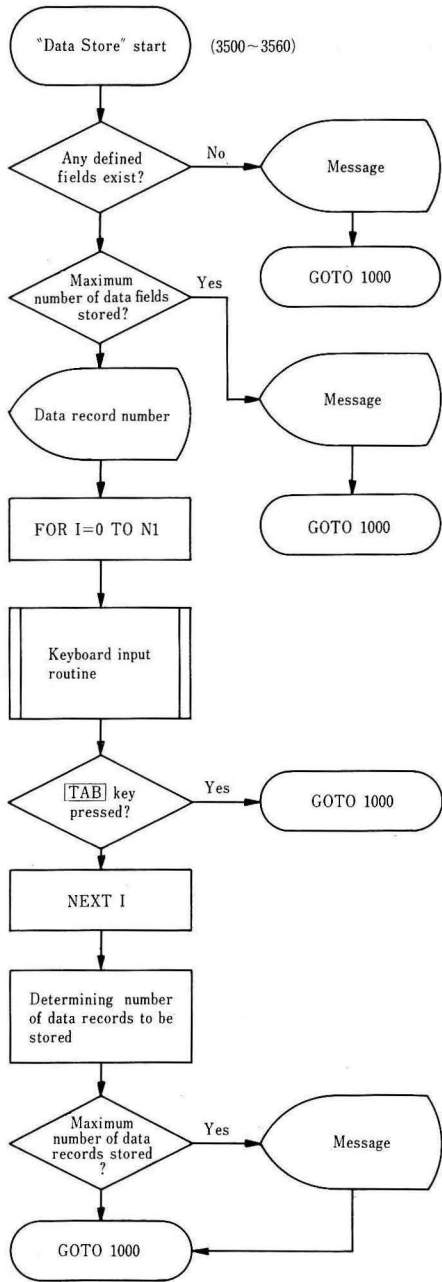


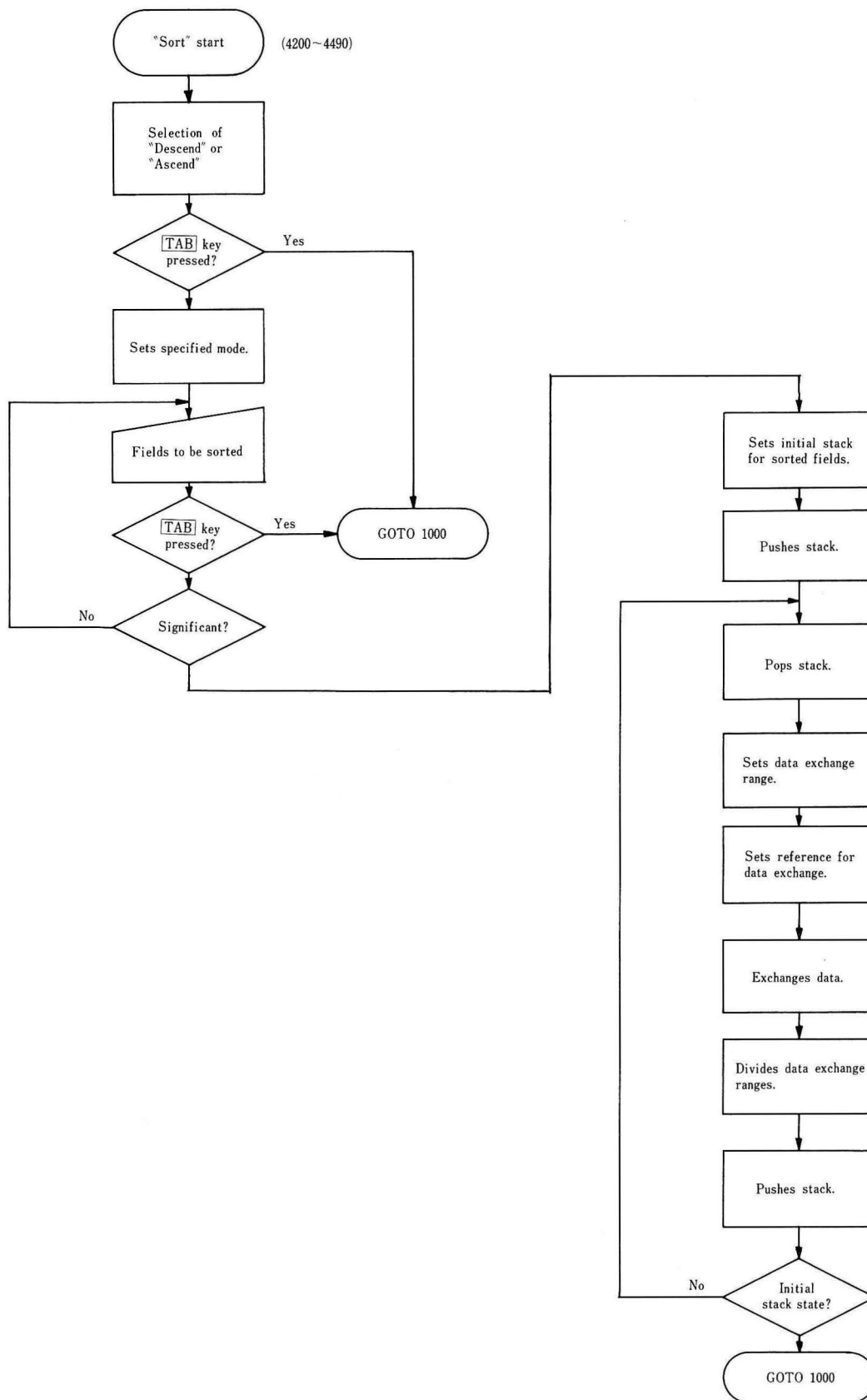
"Storage File"











```

1 CONSOLEC80
2 PRINT".....          This program makes the serial data files 'Variable Data'
3 PRINT".....          and 'DATA-DEC2' which are used in the main program
4 PRINT".....          'DATA PROCESSING MAIN PROGRAM'.
6 PRINT".....          Before executing the main program, the above two files
7 PRINT".....          'Variable Data' and 'DATA-DEC2' must be stored in a
8 PRINT".....          diskette in which the main program is stored.
9 REM
10 DIM F$(2),K$(1),KM(2),LM(2),M$(5),M(8),FR$(4),FF$(4)
20 DIM FM$(6),FS$(4),ER$(14),W4$(1),CP(4),S$(2),O$(4),DT$(2)
30 CP(0)=21:CP(1)=23:CP(2)=24:CP(3)=1:CP(4)=17
40 MD$="JUNFEBMARAPRMAYJUNJULAUUGSEPOCTNOVDEC"
41 O$=" Field  ":O$(0)="["+O$+"A ] + ["+O$+"B ] = ["+O$+"C ]"
42 O$(1)="["+O$+"A ] - ["+O$+"B ] = ["+O$+"C ]"
43 O$(3)="["+O$+"A ] / ["+O$+"B ] = ["+O$+"C ]":O$(4)="Summation of one field"
44 FOR I=1 TO 79:S$(0)=S$(0)+" ":S$(1)=S$(1)+" ":S$(2)=S$(2)+" ":NEXT
45 LM(0)=16:LM(1)=2:LM(2)=1:KM(0)=0:KM(1)=1:KM(2)=1:R$="Storage File"
46 O$(2)="["+O$+"A ] * ["+O$+"B ] = ["+O$+"C ]"
50 W4$(0)="|":GT$="16121212001312"
60 K$(0)="Alphanumeric":K$(1)="Numeric"
70 DT$(0)="Month ....":DT$(1)="DAY .....":DT$(2)="Year ....."
80 M$(0)=" 1.Format ":M$(1)=" 2.Read ":M$(2)=" 3.Write ":M$(3)=" 4.Inform "
90 M$(4)=" 5.Process ":M$(5)=" 6.Input Data"
100 M(0)=12:M(1)=23:M(2)=32:M(3)=42:M(4)=53:M(5)=65:M(6)=12:M(7)=27:M(8)=42
110 F$(0)="Field Name      (Up to 16 characters)"
120 F$(1)="Field Length   (Total Length < 80)"
130 F$(2)="Field Attribute(0=Alphanumeric, 1=Numeric)"
140 FR$(0)=" Main Job : ":FR$(1)=" Subjob : ":FR$(2)=" Select : "
150 FR$(3)=" Error : ":FR$(4)=" Message : "
160 FF$(0)=" Main job : ":FF$(1)=" Subjob : ":FF$(2)=" Select : "
170 FF$(3)=" Error : ":FF$(4)=" Message : "
180 FM$(0)=" 1.Format 2.Read 3.Write 4.Inform 5.Process 6.Data Input"
190 FM$(1)=" 1.Define 2.Clear"
200 FM$(2)=" 1.Disk Read 2.RAM Read"
210 FM$(3)=" 1.To Disk 2.To Printer"
220 FM$(5)=" 1.Operations 2.Operation 3.Sort"
230 FM$(6)=" 1.Data Store 2.Correction"
240 FS$(0)=" Specify one of above files [ 1 to   ]"
250 FS$(1)=" 1.Auto Read 2.Step Read"
260 FS$(2)=" 1.Descent 2.Ascent"
270 FS$(3)=" Specify operational format number."
280 FS$(4)=" Return to main job select with pressing any key."
290 ER$(0)=" Invalid ..... No Data or No File"
300 ER$(1)=" Invalid Data"
310 ER$(2)=" Total length is over 79.":ER$(5)=" File is filled."
320 ER$(3)=" Total length is filled.":ER$(6)=" Impossible ... Alphanumeric Field"
325 ER$(4)=" Fields are filled."
330 ER$(7)=" No stored file"
340 ER$(8)=" Printer is not ready.":ER$(9)=" Printer is in mechanical trouble."
350 ER$(10)=" Print paper is lack.":ER$(11)=" Disk drive is not ready."
360 ER$(12)=" Disk memory capacity is lack.":ER$(13)=" Diskette is not initialized."
370 ER$(14)=" Same file name exists."
380 WOPEN#1,"Variable Data":PRINT#1,K$(0),K$(1),W4$(0),GT$,MD$,R$
390 FORI=0TO2:PRINT#1,S$(I),LM(I),KM(I),DT$(I),F$(I):NEXT
400 FOR I=0 TO 4:PRINT#1,CP(I),O$(I),FR$(I),FF$(I),FS$(I):NEXT
410 FOR I=0 TO 6:PRINT#1,FM$(I):NEXT
420 FOR I=0 TO 5:PRINT#1,M$(I):NEXT
430 FOR I=0 TO 8:PRINT#1,M(I):NEXT
440 FOR I=0 TO 14:PRINT#1,ER$(I):NEXT:CLOSE
500 WOPEN#1,"DATA-DEC2":FORI=0TO378:READ A:PRINT#1,A:NEXT:CLOSE:END
600 DATA 213,42,254,255,58,252,255,95,22,0,25,34,252,255,58,251,255,95,229,25,4
3
610 DATA 34,254,255,58,249,255,183,40,11,62,32,67,35,4,205,122,12,43,16,250,225
620 DATA 1,127,32,58,250,255,183,40,3,1,58,48,62,250,8,123,186,32,31,205,113,8

```

630 DATA 205,176,255,205,214,255,8,61,32,238,205,104,12,254,128,56,4,214,128,24
,2
640 DATA 198,128,205,122,12,24,218,205,138,255,24,226,205,122,12,35,20,201,122,1
83
650 DATA 202,128,255,21,205,116,255,43,201,123,186,202,128,255,205,116,255,35,2
0
660 DATA 201,122,183,202,128,255,123,146,254,0,40,228,197,79,6,0,229,213,205,11
6,255
670 DATA 84,93,27,205,135,12,43,62,32,205,122,12,209,225,43,21,193,201,123,186
680 DATA 40,91,229,42,254,255,205,104,12,254,32,225,32,,79,205,116,255,123,61
690 DATA 146,254,0,40,186,197,71,42,254,255,43,205,104,12,35,205,122,12,43,16,2
45
700 DATA 62,32,205,122,12,193,201,209,209,62,128,18,1,1,0,201,193,205,116,255
710 DATA 42,252,255,75,6,0,209,213,197,205,104,12,18,35,19,13,32,247,193,209,20
1
720 DATA 205,104,12,254,128,216,214,128,205,122,12,201,205,190,14,205,190,14
730 DATA 205,190,14,201,205,113,8,184,56,4,185,218,223,254,245,58,250,255,183
740 DATA 40,56,241,254,43,202,223,254,254,45,202,223,254,254,46,202,223,254
750 DATA 254,32,202,223,254,254,0,200,254,13,40,164,254,27,40,151,254,4,202,229
,254
760 DATA 254,7,202,251,254,254,3,202,240,254,254,8,202,33,255,205,128,255,201
770 DATA 241,24,218,197,6,96,16,254,193,201,1,80,5,17,160,208,33,240,208,205,13
5,12
780 DATA 201,6,160,205,113,8,254,0,32,2,16,247,50,249,255,201

```

1 REM.....          DATA PROCESSING MAIN PROGRAM          .....
2 REM
3 CONSOLEC80:PRINTCHR$(6);:PRINT"Do you need the explanation of this program?"
4 PRINT"If so, press the 'Y' key.":PRINT"The main program is executed with pres
sing the other key."
5 PRINT:PRINT"(Note)  You should use the diskette except the master diske
tte on which          this program is rewritten."
6 PRINT"          Files 'Data Processing', 'Explanation-D.P', 'Variable Data' and
'DATA-DEC2' should be transferred."
7 PRINT"          With executing the program '**String-Machine', 'Variable Data'
and 'DATA-DEC2' will be made."
8 A$="":GETA$:IFA$=""GOTO8
9 IFA$="Y"THENCHAIN"Explanation-D.P"
10 LIMIT$FE7E:KILL:DIMA$(9,201),A(9),B(9),F$(2),K$(1),KM(2),LM(2),M$(5),M(8),FR
$(4),FF$(4),FM$(6),FS$(4),ER$(14),W4$(1),C(9),D(9),CP(4),S$(2)
30 DIME(2),O$(4),DT$(2),X(1,100):A3=1:TEMPO6:P=1:P1=0:ON ERROR GOTO6000
100 ROPEN#1,"Variable Data":INPUT#1,K$(0),K$(1),W4$(0),GT$,MD$,R$:FORI=0TO2:INP
UT#1,S$(I),LM(I),KM(I),DT$(I),F$(I):NEXT
120 FORI=0TO4:INPUT#1,CP(I),O$(I),FR$(I),FF$(I),FS$(I):NEXT:FORI=0TO6:INPUT#1,F
M$(I):NEXT:FORI=0TO5:INPUT#1,M$(I):NEXT
130 FORI=0TO8:INPUT#1,M(I):NEXT:FORI=0TO14:INPUT#1,ER$(I):NEXT:CLOSE
200 X=65150:ROPEN#1,"DATA-DEC2":FORI=0TO378:INPUT#1,A:POKEX+I,A:NEXT:CLOSE
500 PRINTCHR$(6);S$(0):PRINTS$(0):CURSOR0,20:FORI=1TO4:PRINTS$(0):NEXT:PRINTS$(
0);:CURSOR0,0
520 PRINTTAB(13);"*** Data Processing Program ***";TAB(59);"Date : "
700 CURSOR16,17:PRINT"..... Input today's date .....":A7=3:GOSUB7080
710 FORI=0TO2
720 CURSOR27,17:PRINTDT$(I):A4=1:A5=2:IFI=2THENA5=4
730 A6=37:GOSUB7600:IFD1=0GOSUB7070:GOTO700
740 IFIP<1THEN7930
750 ONIGOTO800,820
760 IFIP>126GOTO7930
770 X=IP:GOTO820
800 Y=IP:IF(IP>31)+((X=2)*(IP>29))+(((X=4)+(X=6)+(X=9)+(X=11))*(IP=31))GOTO7930
820 A7=1:GOSUB7080:NEXT
840 DT$=LEFT$(STR$(Y)+" "+MID$(MD$,3*X-2,3)+" "+STR$(IP)+S$(1),12)
850 A$=DT$:GOSUB7900:CURSOR0,0:PRINTTAB(78-LEN(TD$));TD$
1000 GOSUB7070:CURSOR0,1:PRINTS$(0):CURSOR0,21:PRINTFR$(0);FM$(0):PRINT:PRINTS$(
0):PRINTS$(0);
1010 CP=0:GF=0:K=0:JM=0:GOSUB7100:JM=G:IFG3=0GOTO1010
1130 CURSDRM(G-1),21:PRINTM$(G-1):IFJM=4GOTO3000
1160 CURSOR0,23:PRINTFF$(1);FM$(JM):CP=1:GF=0:K=1:GOSUB7100:SJ=G:IFG3=0GOTO1000
1200 CURSDRM(G+5),23:A$=FM$(JM):GOSUB7910:PRINTTD$
1210 ONJMGOTO1500,2000,2500,3000,4000,3500
1250 ER=0
1260 GOSUB7400:GOTO1000
1500 IFSJ=16GOTO1540
1530 GOSUB7920:PRINT"clearing buffer !":FORL1=0TON1:FORL2=0TOP1:A$(L1,L2)=""N
EXTL2,L1:P=1:NF=0:P1=0:N1=0:U=0:GOTO1000
1540 IF(NF>=10)+(U>=79)THENER=5:GOTO1260
1545 Y=NF:A6=37:ZZ=0
1560 FORI=NFTO9:FORJ=0TO2:IF(J=0)*(U=79)GOTO1695
1580 CP=1:X=1:GOSUB7700:PRINT"Information : Input ";F$(J);:A4=KM(J):A5=LM(J)
1585 IFJ=1THENPRINTTAB(60);" [ Remainder = ";79-U;" ]"
1590 IFJ=0THENCURSOR0,17:PRINT" Field : ";A$=STR$(I+1):GOSUB7900:PRINTTD$;" "
1600 CURSOR15,17:PRINTLEFT$(F$(J),15);" =":GOSUB7600:IFD1=0GOTO1700
1610 ONJGOTO1630,1660
1620 A$(I,0)=IP$:GOTO1680
1630 IFIP<=0THENER=1:GOSUB7400:GOTO1590
1640 U=U+IP:IFU>79THENU=U-IP:ER=2:GOSUB7400:GOTO1590
1650 B(I)=IP:GOTO1680
1660 A(I)=IP:IF(IP<>0)*(IP<>1)THENER=1:GOSUB7400:GOTO1580
1680 A7=2:GOSUB7080:NEXTJ
1690 A7=3:GOSUB7080:NF=NF+1:N1=NF-1:NEXTI
1695 X=1:CP=24:GOSUB7700:PRINTFF$(4);"      ";ER$(4);:K=4:CP=2:GF=2:GOSUB7110
1700 IFY=NF:GOTO1000

```

```

1705 IFP=1GOTO1000
1710 FORI=1TOP1:FORJ=YTON1:IFA(J)=OTHENA$(J,I)=LEFT$(S$(1),B(J)):GOTO1740
1730 A$(J,I)=LEFT$(S$(1),B(J)-1)+"0"
1740 NEXTJ,I:GOTO1000
2000 IFSJ=2GOTO2250
2005 N=0:M=0:XOPEN#1,R#:INPUT#1(1),AA:IFAA=OTHENCLOSE:GOTO1250
2020 FORI=1TOAA:CURSOR0,17:IFN=6GOTO2090
2030 IFN<>OGOSUB7940:GOTO2070
2040 PRINT"[ No. ]";TAB(12);"[ File Name ]";TAB(32);"[ Date ]";
2050 PRINTTAB(46);"[ No. of Data ]";TAB(63);"[ No. of Field ]":A7=2:GOSUB7080:G
OSUB7940
2070 CURSOR0,17:PRINTTAB(3);N+1;". ";TAB(10);A#;TAB(31);TD#;TAB(53);P1;TAB(71);N
F:N=N+1
2080 A7=1:GOSUB7080:NEXT
2090 CURSOR50,17:IF(6*M+N)<AATHENPRINT"..... To be continued.":CM=1:GOTO2
110
2100 CM=0:PRINT"..... End"
2110 CP=24:X=1:GOSUB7700:PRINTTAB(13);FS$(0);:CURSOR47,24:PRINTN;:CP=2:G1=1:G2=
N:K=2:GF=3:GOSUB7110:ONG3+1GOTO2140,2150
2130 IFCM=1THENM=M+1:N=0:GOSUB7070:CURSOR0,17:GOTO2030
2140 CLOSE:GOTO1000
2150 I=6*M+6:GOSUB7940:CLOSE:GOSUB7070:GOSUB7920:PRINT"reading !!!"
2180 ROPEN#1,A#:FORI=0TON1:FORJ=0TOP1:INPUT#1,A$(I,J):NEXTJ,I:U=0
2200 FORI=0TON1:INPUT#1,A(I),B(I):U=U+B(I):NEXT:CLOSE:GOTO1000
2250 IFP=1GOTO1250
2252 CURSOR0,24:PRINTFR$(2);FS$(1);:G1=1:G2=2:CP=2:K=2:GF=0:GOSUB7110
2260 IFG3=0GOTO1000
2263 CURSORM(5+G),24:A#=FS$(1):GOSUB7910:PRINTTD#;:GOSUB7450:I=0:SS=G
2280 IFI=P1THENI=0:A7=1:GOSUB7080
2290 I=I+1:CURSOR0,16:PRINT"16. ";A#=STR$(I):GOSUB7900:PRINTTD#
2300 FORJ=0TON1:PRINTA$(J,I);:NEXTJ:PRINTIT$:IFSS=2GOTO2350
2310 GF=1:GG=2:GOSUB7800
2320 ONG3+1GOTO2375,1000,2350,2360
2350 CURSOR70,24:PRINT" STOP ";:GF=2:GOSUB7800
2355 ONG3GOTO1000,2370,2360
2360 SS=G:CURSOR12,24:IFG=2THENPRINT "1.Auto Read 2.Step Read ";:GOTO2367
2365 PRINT" 1.Auto Read 2.Step Read ";
2367 IFSS=2GOTO2350
2370 CURSOR70,24:PRINT" ";
2375 A7=2:GOSUB7080:GOTO2280
2500 IFP=1GOTO1250
2505 IFSJ=2GOTO2650
2510 ZZ=0:CURSOR 0,17:PRINT"Specify file name (Up to 16 characters) : "
2520 A4=0:A5=16:A6=45:GOSUB7010:IFASC(IP#)=128GOTO1000
2540 FORI=1TO16:IFASC(MID$(IP#,I,1))<>32GOTO2555
2550 NEXT:ER=1:GOSUB7400:GOTO2510
2555 GOSUB7080:GOSUB7920:PRINT"writing !!!"
2560 WOPEN#1,IP#:FORI=0TON1:FORJ=0TOP1:PRINT#1,A$(I,J):NEXTJ,I
2580 FORI=0TON1:PRINT#1,A(I),B(I):NEXT:CLOSE:XX#=IP#+DT#+STR$(N1)+STR$(P)
2590 XOPEN#1,R#:INPUT#1(1),AA:AA=AA+1:PRINT#1(1),AA:PRINT#1(AA+1),XX#:CLOSE:GOT
O1000
2650 PRINT/PCHR$(18);"*** Data Processing *** ";DT$
2660 PRINT/P:PRINT/P:FORI=0TON1:PRINT/P:TAB(50);"Field";I+1;"=" ";A$(I,0):NEXT:PR
INT/P:PRINT/P
2700 B=0:W1$="":W2$="":W3$="":W$="":H=0:IF(U+N1)>79THENH=1
2710 FORI=0TON1:C(I)=B+B(I)-INT(B(I)/2)-1:B=B+B(I):D(I)=B-1:IFH=OTHENC(I)=C(I)+
I:D(I)=D(I)+I+1
2720 NEXTI:FORI=0TON1:W$="":FORJ=1TOB(I)-H:W$=W$+"-":NEXTJ:W1$=W1$+W$+"_":W2$=W
2$+W$+"+":W3$=W3$+W$+"-":
2740 NEXTI:W1$=LEFT$(W1$,LEN(W1$)-1)+"-":W2$=LEFT$(W2$,LEN(W2$)-1)+"-":W3$=LEFT
$(W3$,LEN(W3$)-1)+"-":PRINT/PCHR$(17)
2750 PRINT/PW1$:FORI=1TONF:PRINT/P:TAB(C(I-1));STR$(I);TAB(D(I-1));:IFI<>NFTHENP
RINT/PW4$(0);
2760 NEXT:PRINT/P:PRINT/PW2$
2770 FORI=1TOP1:FORJ=0TON1:PRINT/PA$(J,I);:IFJ<>N1THENPRINT/PW4$(H);
2785 NEXTJ:PRINT/P:GF=1:GOSUB7800:IFG3<>0GOTO2840

```

```

2820 IFI=P1THENPRINT/PW3#:GOTO2840
2830 PRINT/PW2#:NEXTI
2840 PRINT/PCHR$(16):GOTO1000
3000 IFNF=0GOTO1250
3005 CURSOR3,17:PRINT"[ Field ]";TAB(20);"[ Name ]";TAB(35);"[ Length ]";
3010 PRINTTAB(50);"[ Attribution ]":A7=2:GOSUB7080:FORI=1TO10:CURSOR0,17
3030 PRINTTAB(10-LEN(STR$(I)));STR$(I);". ";TAB(16);:IFI>NFGOTO3070
3040 PRINTA$(I-1,0);TAB(40-LEN(STR$(B(I-1)))));STR$(B(I-1));TAB(50);K$(A(I-1))
3070 A7=0:GOSUB7080:NEXTI:CURSOR48,17:PRINT"[No. of Store Data =";P1;" ]"
3090 X=1:CP=24:GOSUB7700:PRINTFF$(4);" ";FS$(4);:K=4:CP=2:GF=0:G1=-48:G2=207:G
OSUB 7110:GOTO1000
3500 IFNF=0GOTO1250
3502 IFSJ=2GOTO3600
3503 IFP=201THENER=5:GOTO1120
3505 GOSUB7450
3510 CURSOR0,16:PRINT"No. ";P:CURSOR0,18:PRINTIT#
3520 FORI=0TON1:GOSUB7500
3530 IFD1=0GOTO1000
3540 NEXTI:P=P+1:P1=P-1
3550 IFP=201THENX=1:CP=24:GOSUB7700:PRINTFF$(4);" ";ER$(5);:K=4:CP=2:GF=2:GOS
UB7110:GOTO1000
3560 A7=4:GOSUB7080:GOTO3510
3600 CURSOR0,17:PRINT"Specify correction item number (Within";P1;" ) : "
3610 A4=1:A5=LEN(STR$(P)):A6=45:GOSUB7600
3620 IFD1=0GOTO1000
3630 IF (IP>P1)+(IP<1)GOSUB7650:GOTO3610
3640 A7=3:GOSUB7080:J=IP:GOSUB7450
3650 CURSOR0,16:PRINT"No. ";IP:FORI=0TON1:PRINTA$(I,J);:NEXT:PRINT:PRINTIT#
3720 A3=0:B=P:P=J:FORI=0TON1:GOSUB7500:IFD1=0GOTO3740
3730 NEXT
3740 A3=1:P=B:A7=4:GOSUB7080:GOTO3600
4000 IFP=1GOTO1250
4005 DNSJ-1GOTO4500,4200
4007 CURSOR0,17:PRINT"Specify first deletion item number (Within";P1;" ) :":A5=L
EN(STR$(P1))
4010 A6=50:A4=1:GOSUB7600:IFD1=0GOTO1000
4020 IF (IP<1)+(IP>P1)GOSUB7650:GOTO4010
4025 NN=IP:A7=2:GOSUB7080
4030 CURSOR0,17:PRINT"Specify number of deletion item :":A6=50:GOSUB7600
4040 IFD1=0GOTO1000
4050 IFIP<1GOSUB7650:GOTO4030
4060 A7=5:GOSUB7080:GOSUB7920:PRINT"deletion !!"
4080 IF (NN+IP) >=PTHENP=NN:P1=P-1:GOTO4120
4090 PP=P-NN-IP:P=P-IP:P1=P-1
4100 FORJ=0TON1:FORI=0TOPP-1:A$(J,NN+I)=A$(J,NN+IP+I):NEXTI,J
4120 GOSUB7070:GOTO4000
4200 CURSOR0,24:PRINTFF$(2);FS$(2);:GF=0:K=2:CP=2:G1=1:G2=2:GOSUB7110
4220 IFG3=0GOTO1000
4230 CURSORM(G+5),24:A#=FS$(2):GOSUB7910:PRINTTD#;
4240 CURSOR0,17:PRINT"Specify sort field (within";NF;" ) :":A4=1:A5=2:A6=40:
GOSUB7600
4250 IFD1=0GOTO1000
4252 IF (IP<1)+(IP>NF)GOSUB7650:GOTO4240
4254 ST=IP-1:IFA(ST)=0THENER=6:GOSUB7400:CURSOR0,24:PRINTFF$(2);FS$(2);:GOTO423
0
4256 A7=5:GOSUB7080:GOSUB7920:PRINT"sorting !!":Y1=0:IFP1=1GOTO1000
4300 Y2=1:Y3=P1:GOSUB4480
4310 GOSUB4490:I=Y2:J=Y3:X=VAL(A$(ST,INT(Y2+Y3)/2))
4315 IFG=1GOTO4350
4320 FORI=1TOY3:IFVAL(A$(ST,I))<XTHENNEXT
4330 FORJ=JTOY2STEP-1:IFVAL(A$(ST,J))>XTHENNEXT
4340 GOTO4370
4350 FORI=1TOY3:IFVAL(A$(ST,I))>XTHENNEXT
4360 FORJ=JTOY2STEP-1:IFVAL(A$(ST,J))<XTHENNEXT
4370 IFI>JGOTO4400
4380 FORII=0TON1:CH#=A$(II,J):A$(II,J)=A$(II,I):A$(II,I)=CH#:NEXTII

```

```

4390 I=I+1:J=J-1:GOTO4315
4400 Y4=Y3:IFY2<JTHENY3=J:GOSUB4480
4410 Y3=Y4:IFI<Y3THENY2=I:GOSUB4480
4420 IFY1<>0GOTO4310
4430 GOTO1000
4480 X(0,Y1)=Y2:X(1,Y1)=Y3:Y1=Y1+1:RETURN
4490 Y1=Y1-1:Y2=X(0,Y1):Y3=X(1,Y1):RETURN
4500 ZZ=0:CURSOR14,17:PRINT"**** Format of Arithmetic Operation ****":A7=2:GO
SUB7080
4510 FORI=0TO4:CURSOR10,17:PRINT "[";I+1;" ] ";0$(I):A7=1:GOSUB7080:NEXT
4520 CP=24:X=1:GOSUB7700:PRINTFR$(2);FS$(3);:G1=1:G2=5:GF=0:K=2:CP=2:GOSUB7110
4540 IFG3=0GOTO1000
4550 OP=G:GOSUB7070:IFG=5GOTO4900
4553 GOSUB7070:CURSOR0,17:PRINT0$(G-1):A7=2:GOSUB7080
4555 A$=" OperandOperator Result":FORI=0TO2
4560 CURSOR20,17:PRINTMID$(A$,8*I+1,8);" Field ";CHR$(65+I);"="
4570 A4=1:A5=1:A6=40:GOSUB7600:IFD1=0GOTO1000
4580 IF(IP<0)+(IP>NF)GOSUB7650:GOTO4560
4600 IFA(IP-1)=0THENER=6:GOSUB7400:GOTO4560
4610 E(I)=IP-1:A7=1:GOSUB7080:NEXT
4620 A7=2:GOSUB7080:GOSUB7920:PRINT"calculation !!":ONOP-1GOTO4650,4660,4670,5
020
4640 FORI=1TOP1:Z=VAL(A$(E(0),I))+VAL(A$(E(1),I)):GOTO4800
4650 FORI=1TOP1:Z=VAL(A$(E(0),I))-VAL(A$(E(1),I)):GOTO4800
4660 FORI=1TOP1:Z=VAL(A$(E(0),I))*VAL(A$(E(1),I)):GOTO4800
4670 FORI=1TOP1:IFVAL(A$(E(1),I)=0THENA$(E(2),I)=LEFT$(S$(2),B(E(2))):GOTO4820
4680 Z=VAL(A$(E(0),I))/VAL(A$(E(1),I))
4800 Z=INT(Z*10+0.5)/10:A$(E(2),I)=RIGHT$(S$(1)+STR$(Z),B(E(2)))
4810 IFLEN(STR$(Z))>B(E(2))THENA$(E(2),I)=LEFT$(S$(2),B(E(2)))
4820 NEXT:GOTO1000
4900 CURSOR0,17:PRINT0$(OP-1);" ";TAB(29);"Field ="
4910 A4=1:A5=2:A6=37:GOSUB7600:IFD1=0GOTO1000
5000 IF(IP<0)+(IP>NF)GOSUB7650:GOTO4910
5010 GOTO4620
5020 J=IP-1:LK=0:FORI=1TOP1:LK=LK+VAL(A$(J,I)):NEXT:A$(J,151)=STR$(LK)
5030 GOSUB7070:CURSOR0,17:PRINT"Field";IP;" ";TAB(14);"SAM=";LK;TAB(40);"AVER
AGE=";INT(LK/P1*100)/100:GOTO3090
6000 IF((ERL=100)+(ERL=200))*((ERN=50)+(ERN=40))THENCURSOR0,17:PRINT"Data file
does not exist. Otherwise, disk is not ready.":KILL:END
6005 IF(ERL=7550)+(ERL=7620)THENIP$="0":RESUME
6010 IF(ERL=2560)*(ERN=42)THENER=14:GOSUB7400:KILL:RESUME2510
6020 IF(ERN=65)+(ERN=66)+(ERN=67)THENER=ERN-57:GOTO6060
6030 IFERN=50THENER=11:GOTO6060
6040 IF(ERN=53)+(ERN=54)THENER=ERN-41:GOTO6060
6050 GOSUB7080:CURSOR0,17:PRINT"Resumption impossible":KILL:END
6060 GOSUB7420:A7=3:GOSUB7080:CURSOR0,17:IFERN=53THENPRINT"Replace with new dis
kette.":A7=3:GOSUB7080:CURSOR0,17
6065 PRINT"Press any key after proper procedure."
6070 GOSUB7780:IFG=-48GOTO6070
6080 GOSUB7070:CURSOR33,17:PRINT"Resumption !!":KILL:IFERN<>53THENRESUME
6090 XOPEN#1,R#:INPUT#1(1),AA:CLOSE:RESUME2560
7000 A6=0:IFI=0GOTO7020
7005 FORI1=0TOI-1:A6=A6+B(I1):NEXT:GOTO7020
7010 POKE$FFFA,A4:POKE$FFFB,A5:GOTO7040
7020 POKE$FFFA,A(I):POKE$FFFB,B(I)
7040 POKE$FFFE,$50:POKE$FFFF,$D5:POKE$FFF9,A3:POKE$FFFC,A6
7050 ROPEN#1,USR($FE7E):INPUT#1,IP#:CLOSE:RETURN
7070 A7=16
7080 FORI1=0TOA7:USR($FFDD):NEXT:RETURN
7100 G1=VAL(MID$(GT$,2*JM+1,1)):G2=VAL(MID$(GT$,2*JM+2,1))
7110 R=0:RV=-1:VR=0
7120 CURSOR0,CP(CP):IFGF=2GOTO7170
7130 GOSUB7780:IFG=-48GOTO7170
7140 IFG=-21THENG3=0:GOTO7240
7145 IFGF=3THENIFG=-16THENG3=2:GOTO7240
7150 IF(G>G1-1)*(G<G2+1)THENG3=1:GOTO7240

```



```

7160 GOSUB7650
7170 IFGF=1GOTO7120
7190 IF (GF=2) * (VR=50) GOTO7240
7200 R=0:RV=-RV:VR=VR+1:IFRV<0THENPRINTFR$(K);:GOTO7120
7210 PRINTFF$(K);:GOTO7120
7240 IFGF<>1THENPRINTFF$(K);
7250 RETURN
7400 CP=24:X=1:GOSUB7700:PRINTFF$(3);ER$(ER);:GOSUB7650
7410 CP=2:K=3:GF=2:GOSUB7110:CP=24:X=ZZ:GOSUB7700:RETURN
7420 GOSUB7070:CURSOR0,17:PRINTFF$(3);ER$(ER):GOSUB7650:CP=4:K=3:GF=2:GOSUB7110
:RETURN
7450 IT$="":FORII=0TON1:IT$=IT$+"^"
7455 IFB(II)>1THENFORJJ=1TOB(II)-1:IT$=IT$+"-":NEXTJJ
7460 NEXTII:RETURN
7500 CP=1:X=1:GOSUB7700:PRINT"Field";I+1;TAB(15);A$(I,0);TAB(40);K$(A(I));
7510 PRINTTAB(60);"Length=";B(I):GOSUB7000:IFASC(IP$)=128THEND1=0:RETURN
7540 IFA(I)=0GOTO7580
7550 IP$=S$(1)+STR$(VAL(IP$)):IP$=RIGHT$(IP$,B(I))
7580 A$(I,P)=IP$:D1=1:RETURN
7600 GOSUB7010:IFASC(IP$)=128THEND1=0:RETURN
7610 IFA4=0THEND1=1:RETURN
7620 IP=INT(VAL(IP$)):IP$=STR$(IP):D1=2:RETURN
7650 MUSIC"+BOR+BR+B":RETURN
7700 CURSOR0,CP:PRINTS$(X);:CURSOR0,CP:RETURN
7780 USR($FFEA):G=PEEK($FFF9)-48:RETURN
7800 GOSUB7780:IFG=-48THENONGFGOTO7850,7800
7810 IFG=-21THENG3=1:RETURN
7820 IFG=-16THEN G3=2:RETURN
7830 IF(G>0)*(G<66+1)THENG3=3:RETURN
7840 GOSUB7650:GOTO7800
7850 G3=0:RETURN
7900 TD$="":FORII=1TOLEN(A$):TD$=TD$+CHR$(ASC(MID$(A$,II,1))+128):NEXT:RETURN
7910 TD$="":FORI=1TO14:TD$=TD$+CHR$(ASC(MID$(A$, (G-1)*15+I,1))-128):NEXT:RETURN
7920 CURSOR15,17:PRINT"Wait a minute !! Under ";:RETURN
7930 GOSUB7650:GOTO720
7940 INPUT#1(I+1),XX$:A$=LEFT$(XX$,16):TD$=MID$(XX$,17,12):N1=VAL(MID$(XX$,29,1
)):P=VAL(MID$(XX$,30,3)):NF=N1+1:P1=P-1:RETURN

```

Chapter 4

Programming Instructions

This chapter summarizes all commands, statements, operators and symbols of the DISK BASIC interpreter SB-6510.

4.1 List of DISK BASIC interpreter SB-6510 commands, statements and functions

4.1.1 Commands

DIR	DIR FDd	<p>Displays the file directory of the diskette in drive d (d=1~4). The contents of the directory are as follows:</p> <ol style="list-style-type: none"> 1) Volume number 2) Number of unused sectors 3) Mode, lock condition and file name of each file on the diskette <p>Note: When a directory is listed on the CRT, the display is fixed and the cursor appears when the frame is filled.</p> <p>To display the next frame of the directory, press the CR key. Other command may be executed once the display is fixed.</p>
	DIR FD3	<p>Displays the files directory of the diskette in drive 3. When a DIR FDd command is executed, the system stores the drive number so that it may be omitted (if the same drive is specified) for the direct execution instructions and file access instructions explained below.</p>
	DIR	<p>Displays the file directory of the diskette in the active drive which is last specified in a DIR FDd command.</p>
DIR/P	DIR FDd/P	<p>Prints the file directory of the diskette in drive d on the line printer.</p>
LOAD	LOAD "A"	<p>Loads BASIC text (BTX) assigned the file name "A" from the diskette in the active drive into the text area.</p>
	LOAD FD2@10 "A"	<p>Loads the BASIC text assigned the file name "A" from volume 10 in drive 2 into the text area.</p>
	LIMIT \$D000: LOAD "B"	<p>To load a machine language program file (OBJ) to be linked with a BASIC text, the BASIC area of memory must be partitioned from the machine language area by the LIMIT statement.</p>
LOAD/T	LOAD/T "C"	<p>Loads the BASIC text assigned the file name "C" from the cassette tape into the text area.</p> <p>Note: When a LOAD command or a LOAD/T command is executed for a BASIC text file, the text area is cleared of any programs previously stored.</p>
SAVE	SAVE "D"	<p>Assigns the file name "D" to the BASIC text in the text area and stores it on the diskette in the active drive. The text is stored in the BTX file mode.</p>

SAVE/T	SAVE/T "E"	Assigns the file name "E" to the BASIC text in the text area and automatically stores it on the cassette tape.
RUN	RUN	Executes the BASIC text in the text area from the top. Note: The RUN command clears all variables (fills them with 0 or null string) before running text.
	RUN 1000	Executes the BASIC text starting at line number 1000.
	RUN "F" ↑ (BTX)	Loads the BASIC text assigned the file name "F" from the diskette in the active drive and executes it from its beginning.
	RUN FD3@7 "G" ↑ (OBJ)	Loads machine language program assigned the file name "G" from the diskette of volume 7 in drive 3, and then executes the program starting at the start address. In such cases, system control is transferred from the BASIC interpreter to the machine language program.
VERIFY	VERIFY "H"	This command automatically compares the program contained in the BASIC text area with its equivalent text assigned the file name "H" in the cassette tape file.
AUTO	AUTO	Automatically generates and assigns line numbers 10, 20, 30 during creation.
	AUTO 200, 20	Automatically generates line numbers at intervals 20 starting at line 200. 200, 220, 240 An AUTO command is terminated by pressing the BREAK key.
LIST	LIST	Displays all lines of BASIC text currently contained in the text area.
	LIST -500	Displays all lines of BASIC text up through line 500.
LIST/P	LIST/P	Prints out all lines contained in the BASIC text area on the line printer.
NEW	NEW	Clears the text area and variable area. Further, disestablishes the machine language program area set by a LIMIT statement by removing the partition.
CONT	CONT	Continues program execution which was halted by a STOP statement or the BREAK key, starting at the statement following the STOP statement or the statement halted by the BREAK key.
MON	MON	Transfers system control from the BASIC interpreter to the MONITOR. (To transfer system control from the MONITOR to the BASIC interpreter, execute monitor command J.)

BOOT	BOOT	Activates the MZ-80B system initial program loader.
KLIST	KLIST	Displays a complete list of string definitions for special function keys, thereby enabling you to determine how individual special function keys are defined.

4.1.2 File control statements

LOCK	LOCK "ABC"	Locks file "ABC" on the diskette in the active drive.
	LOCK FD4@7 "ABC"	Locks file "ABC" on the diskette (whose volume number is 7), in drive 4. The locked file cannot be updated or deleted. When the file directory is listed, the locked file name is indicated with an asterisk.
UNLOCK	UNLOCK "ABC"	Unlocks file "ABC" on the diskette in the active drive.
	100 UNLOCK FD1 "A"	Unlocks file "A" on the diskette in drive 1. (This is an example of a statement used in a program.)
RENAME	RENAME "A", "B"	Changes the name of file "A" on the diskette in the active drive to "B".
DELETE	DELETE "A"	Deletes file "A" from the diskette in the active drive.
CHAIN	CHAIN FD2@7 "TEXT B"	Chains the program in the BASIC text area to BASIC program "TEXT B" on the diskette volume 7 in drive 2. That is, program "TEXT B" is loaded in the BASIC text area and program execution is started at its beginning. Before the text is loaded, the BASIC text area is cleared but all variable values and contents of user functions are given to program "TEXT B". The CHAIN statement has the same function as GOTO "filename".
	CHAIN "TEXT B"	Chains the program in the BASIC text area to program "TEXT B" on the diskette in the active drive.
SWAP	SWAP FD2@7 "TEXT S-R"	Swaps the current program for BASIC program "TEXT S-R" on diskette volume 7 in drive 2. The current program text is saved on the diskette in the drive specified in the last DIR FdD command, then program "TEXT S-R" is loaded into the BASIC text area and is executed from its beginning. When the swapped program is finished, the saved program is loaded again and program execution is started at the statement following the SWAP statement. The values of variables and the contents of user functions are transferred between the two programs. No SWAP statement can be used in a swapped program. The SWAP statement has the same function as GOSUB "filename".

4.1.3 BSD (BASIC Sequential access Data file) control statements

WOPEN #	WOPEN #3, FD2@7, "SEQ DATA 1"	Defines the file name of a BSD (BASIC sequential access data file) to be created as "SEQ DATA 1" and opens it with logical number 3 assigned on diskette volume 7 in drive 2. For WOPEN # statements including a USR function operand, see page 78.
PRINT #	PRINT #3, A, AS	Writes the contents of variable A and string variable AS in order in the BSD assigned logical number 3 which was opened by a WOPEN # statement. (In writing data, 256 bytes are treated as a unit.)
CLOSE #	CLOSE #3 (corresponding to WOPEN #)	Closes the BSD assigned logical number 3 which was opened by the WOPEN #3 statement. By closing the BSD, the BSD which has the file name defined in the WOPEN # statement is created on the specified diskette, and the logical number assigned is made undefined.
KILL #	KILL #	Kills the BSD assigned logical number 3 by the WOPEN # statement. Logical number 3 is made undefined.
ROPEN #	ROPEN #4, FD2@7, "SEQ DATA 1"	Opens BSD "SEQ DATA 1" on diskette volume 7 in drive 2 with logical number 3 assigned to read data in BSD. For ROPEN # statements including a USR function, see page 79.
INPUT #	INPUT #4 A(1), B\$	Reads data sequentially from the beginning of the BSD assigned logical number 4 which was opened by the ROPEN # statement and substitutes numerical data into array element A(1) and string data into string variable B\$.
CLOSE #	CLOSE #4 (corresponding to ROPEN #)	Close the BSD assigned file number 4 and makes the file number undefined.

4.1.4 BRD (BASIC Random access Data file) control statements

XOPEN #	XOPEN #5, FD3@18, "DATA R1"	Generally, XOPEN# statement opens a BRD for writing and reading data (Cross open). This statement cross-opens BRD "DATA R1" on diskette volume 18 in drive 3 with logical number 5 assigned or, if the file does not exist on the diskette, cross-opens a BRD by defining its file name as "DATA R1" to create it on the diskette with logical number 5 assigned.
PRINT #()	PRINT #5(11), R(11)	Writes the content of linear array element R(11) on field 11 of the BRD assigned logical number 5 which was opened by the XOPEN # statement.

PRINT # ()	PRINT #5(20), AR\$, ASS\$	Writes the contents of string variables AR\$ and ASS\$ on field 20 and field 21 of the BRD assigned logical number 5, respectively. All BRD fields have a fixed length of 32 bytes and, if the length of string variable exceeds 32 bytes, the excess part is discarded.
INPUT # ()	INPUT #5(21), R\$	Reads the content of field 21 of the BRD assigned logical number 5 which was opened by the XOPEN # statement into string variable R\$.
	INPUT #5(11), A(11), A\$(12)	Reads the contents of field 11 and field 12 of the BRD assigned logical number 5 into linear numeric array element A(11) and linear string array element A\$(12), respectively.
CLOSE #	CLOSE #5	Closes the BRD assigned logical number 5 which was opened by the corresponding XOPEN # statement.
	CLOSE	Closes all open files.
KILL #	KILL	Kills all open files.
IF EOF (#)	10 IF EOF (#5) THEN 700	Transfers program control to the routine starting to line number 700 if an EOF (End of File) is detected when an INPUT # statement is executed against a BSD or a BRD.

4.1.5 Error processing statements

ON ERROR GOTO	ON ERROR GOTO 1000	Declares that the number of the line to which program execution is to be moved, if an error occurs is 1000.
IF ERN	IF ERN=44 THEN 1050	Jumps to the statement on line number 1050 if the error number is 44.
IF ERL	IF ERN=350 THEN 1090	Jumps to the statement on line number 1090 if the error line number is 350.
	IF (ERN=53)*(ERL=700) THEN END	Terminates the program if the error number is 53 and the error line number is 700. With DISK-BASIC, the error number and error line number are set in special variables ERN and ERL, respectively, if an error occurs during program execution.
RESUME		Returns program execution to the main program after correction of an error.
	650 RESUME	Returns program execution to the statement in which the error occurred.

	700 RESUME NEXT	Returns program execution to the statement just after the one in which the error occurred.
	750 RESUME 400	Returns program execution to line number 400.
	800 RESUME 0	Returns program execution to the beginning of the program.

4.1.6 Cassette data file input/output statements

WOPEN/T	10 WOPEN/T "DATA-1"	Defines the file name of a cassette data file to be created as "DATA-1" and opens.
PRINT/T	20 PRINT/T A, A\$	Writes the contents of variable A and string variable A\$ in order in the cassette data file which was opened by a WOPEN/T statement.
CLOSE/T	30 CLOSE/T	Closes the cassette data file which was opened by a WOPEN/T statement.
ROPEN/T	110 ROPEN/T "DATA-2"	Opens the cassette data file specified with file name "DATA-2".
INPUT/T	120 INPUT/T B, B\$	Reads data sequentially from the beginning of the cassette data file which was opened by the ROPEN/T statement and substitutes numerical data into variable B and string data into string variable B\$ respectively.
CLOSE/T	130 CLOSE/T	Closes the cassette data file which was opened by a ROPEN/T statement.

4.1.7 Assignment statement

LET	<LET> A = X + 3	Substitutes X + 3 into numeric variable A. LET may be omitted.
------------	-----------------	--

4.1.8 Input/output statements

PRINT	10 PRINT A	Displays the numeric value of A on the CRT screen.
	? A\$	Displays the character string of variable A\$ on the CRT screen.
	100 PRINT A; A\$, B; B\$	Combinations of numeric variables and string variables can be specified in a PRINT statement. When a semicolon is used as the separator, no space is displayed between the data strings. When a colon is used, variable data to the right of the colon is displayed from the next tab set position. (A tab is set every 10 character positions.)

	110 PRINT "COST ="; CS	Displays the string between double quotation marks as is, and CS.
	120 PRINT	Performs a new line operation (i.e., advances the cursor one line).
INPUT	10 INPUT A	Obtains numeric data for variable A from the keyboard.
	20 INPUT A\$	Obtains string data for string variable A\$ from the keyboard.
	30 INPUT "VALUE?"; D	Displays "VALUE?" on the screen before obtaining data from the keyboard. A semicolon separates the string from the variable.
	40 INPUT X, X\$, Y, Y\$	Numeric variables and string variables can be used in combination by separating them from each other with a comma. The types of data entered from the keyboard must be the same as those of the corresponding variables.
GET	10 GET N	Obtains a numeral for variable N from the keyboard. When no key is pressed, zero is substituted into N.
	20 GET K\$	Obtains a character for variable K\$ from the keyboard. When no key is pressed, a null is substituted into K\$.
READ~DATA		Substitutes constants specified in the DATA statement into the corresponding variables specified in the READ statement. The corresponding constant and variable must be of the same data type.
	10 READ A, B, C 1010 DATA 25, -0.5, 500	In READ and DATA statements at left, values of 25, -0.5 and 500 are substitutes for variables A, B and C, respectively.
	10 READ H\$, H, S\$, S 30 DATA HEART, 3 35 DATA SPADE, 11	In the example at left, the first string constant of the DATA statement on line number 10 is substituted into the first variable of the READ statement; that is; "HEART" is substituted into H\$. Then, numeric constant 3 is substituted into numeric variable H, and so on.
RESTORE		With a RESTORE statement, data in the following DATA statement which has already been read by preceding READ statements can be re-read from the beginning by the following READ statements.
	10 READ A, B, C 20 RESTORE 30 READ D, E 100 DATA 3, 6, 9, 12, 15	The READ statement on line number 10 substitutes 3, 6 and 9 into variables A, B and C, respectively. Because of the RESTORE statement, the READ statement on line number 30 substitutes not 12 and 15, but 3 and 6 again into D and E, respectively.

4.1.9 Loop statement

FOR ~ TO NEXT	10 FOR A=1 TO 10 20 PRINT A 30 NEXT A	The statement on line number 10 specifies that the value of variable A is varied from 1 to 10 in increments of one. The initial value of A is 1. The statement on line number 20 displays the value of A. The statement on line number 30 increments the value of A by one and returns program execution to the statement on line number 10. Thus, the loop is repeated until the value of A becomes 10. (After the specified number of loops has been completed, the value of A is 11.)
	10 FOR B=2 TO 8 STEP 3 20 PRINT B^2 30 NEXT	The statement on line number 10 specifies that the value of variable B is varied from 2 to 8 in increments of 3. The value of STEP may be made negative to decrement the value of B.
	10 FOR A=1 TO 3 20 FOR B=10 TO 30 30 PRINT A, B 40 NEXT B 50 NEXT A	The FOR-NEXT loop for variable A includes the FOR-NEXT loop for variable B. As is shown in this example, FOR-NEXT loops can be enclosed in other FOR-NEXT loops at different levels. Lower level loops must be completed within higher level loops. The maximum number of levels of FOR-NEXT loops is 16.
	60 NEXT B, A 70 NEXT A, B	In substitution for NEXT statement at line numbers 40 and 50, a statement at line number 60 shown at left can be used. However, statement at line number 70 cannot be used, causing an error to occur.

4.1.10 Branch statements

GOTO	100 GOTO 200	Jumps to the statement on line number 200.
GOSUB ~ RETURN	100 GOSUB 700 800 RETURN	Calls the subroutine starting on line number 700. At the end of subroutine, program execution returns to the statement following the corresponding GOSUB statement.
IF ~ THEN	10 IF A>20 THEN 200	Jumps to the statement on line number 200 when the value of variable A is more than 20; otherwise the next line is executed.
	50 IF B<3 THEN B=B+3	Substitutes B+3 into variable B when the value of B is less than 3; otherwise the next line is executed.
IF ~ GOTO	100 IF A>=B THEN 10	Jumps to the statement on line number 10 when the value of variable A is equal to or greater than the value of B; otherwise the next line is executed.

IF ~ GOSUB	30 IF A=B*2 GOSUB 90	Jumps to the subroutine starting on line number 700 when the value of variable A is twice the value of B; otherwise the next statement is executed. (When other statements follow a conditional statement on the same line and the conditions are not satisfied, those following an ON statement are executed sequentially, but those following an IF statement are ignored and the statement on the next line is executed.)
ON ~ GOTO	50 ON A GOTO 70, 80, 90	Jumps to the statement on line number 70 when the value of variable A is 1, to the statement on line number 80 when it is 2 and to the statement on line number 90 when it is 3. When the value of A is 0 or more than 3, the next statement is executed. This statement has the same function as the INT function, so that when the value of A is 2.7, program execution jumps to the statement on line number 80.
ON ~ GOSUB	90 ON A GOSUB 700, 800	Jumps to the subroutine on line number 700 when the value of variable A is 1 and jumps to the subroutine on line number 800 when it is 2.


4.1.11 Definition statements

DIM		When an array is used, the number of array elements must be declared with a DIM statement. The number of elements ranges from 0 to 255.
	10 DIM A(20)	Declares that 21 array elements, A(0) through A(20), are used for one-dimensional numeric array A(n).
	20 DIM B(79, 79)	Declares that 6400 array elements, B(0, 0) through B(79, 79), are used for two-dimensional numeric array B(m, n).
	30 DIM C1\$(10)	Declares that 11 array elements, C1\$(0) through C1\$(10), are used for one-dimensional string array C1\$(n).
	40 DIM K\$(7, 5)	Declares that 48 array elements, K\$(0, 0) through K\$(7, 5), are used for two-dimensional string array K\$(m, n).
DEF FN	100 DEF FNA(X)=X^2-X 110 DEF FNB(X)=LOG(X) +1 120 DEF FNZ(Y)=LN(Y)	A DEF FN statement defines a function. The statement on line number 100 defines FNA(X) as $X^2 - X$. The statement on line number 110 defines FNB(X) as $\log_{10} X + 1$ and the statement on line number 120 defines FNZ(Y) as $\log_e Y$. The number of variables included in the function must be 1.
DEF KEY	15 DEF KEY(1)=LIST 25 DEF KEY(2)=LOAD! RUN	A DEF KEY statement defines a function for any of the ten special function keys. The statement on line number 15 defines special function key 1 as LIST. The statement on line number 25 defines special function key 2 as the multi-command LOAD: RUN

4.1.12 Comment and control statements

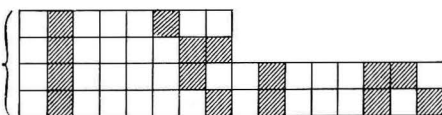
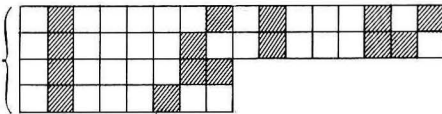
REM	200 REM JOB-1	Comment statement (not executed).
STOP	850 STOP	Stops program execution and awaits a command entry. When a CONT command is entered, program execution is continued.
END	1999 END	Declares the end of a program. Although the program is stopped, the following program is executed if a CONT command is entered.
CLR	300 CLR	Clears all variables and arrays, that is, fills all numeric variables and arrays with zeros and all string variables and arrays with nulls.
CURSOR	50 CURSOR 25, 15 60 PRINT "ABC"	The CURSOR command moves the cursor to any position on the screen. The first operand represents the horizontal location of the destination, and must be between 0 and 39 in 40-character mode, and must be between 0 and 79 in 80-character mode. The second operand represents the vertical location of the destination and must be between 0 and 24. The left example displays "ABC" starting at location (25, 15) (the 26th position from the left side and the 16th position from the top).
CSRH		System variable indicating the X-coordinate (horizontal location) of the cursor.
CSRV		System variable indicating the Y-coordinate (vertical location) of the cursor.
CONSOLE	10 CONSOLE S10, 20	Sets the scrolling area to lines 10 through 20.
	20 CONSOLE C80	Sets the display in the 80 characters/line mode.
	30 CONSOLE C40	Sets the display in the 40 characters/line mode.
	40 CONSOLE R	Sets the display in the reverse mode.
	50 CONSOLE N	Sets the display in the normal mode.
CHANGE	10 CHANGE	Reverses the function of the SHIFT key concerned with alphabetic keys.
REW	710 REW	Rewinds the cassette tape.
FAST	720 FAST	Fast-forwards the cassette tape.
SIZE	? SIZE	Displays the amount of unused memory area in bytes.
TIS	100 TIS = "102030"	Sets the built-in clock to 10:20:30 AM. Data between the double quotation marks must be numerals.

4.1.13 Music control statements

MUSIC		The MUSIC statement generates a melody from the speaker according to the melody string data enclosed in quotation marks or string variables at the tempo specified by the TEMPO statement. The TEMPO statement on line number 300 specifies tempo 7. The MUSIC statement on line number 310 generates a melody consisting of D, E, F sharp, G and A. Each note is a quarter note. When the TEMPO statement is omitted, default tempo is set.
TEMPO		
	300 TEMPO 7	In this example, the melody is divided into 3 parts and substituted in 3 string variables. The following melody is generated from the speaker at tempo 4.
	310 MUSIC "DE#FGA"	
	300 M1\$ = "C3EG + C"	
	310 M2\$ = "BGD - G"	
	320 M3\$ = "C8R5"	
	330 MUSIC M1\$,M2\$,M3\$	

4.1.14 Graphic control statements

GRAPH	10 GRAPH I1	Places graphic area 1 in the input mode. (That is, data are to be transferred to graphic area 1.)
	20 GRAPH O1	Places graphic area 1 in the output mode.
	30 GRAPH O2	Places graphic area 2 in the output mode.
	40 GRAPH O12	Places graphic areas 1 and 2 in the output mode.
	50 GRAPH O0	Resets the graphic output.
	60 GRAPH C	Clears graphic area that is in the input mode.
	70 GRAPH F	Fills graphic area that is in the input mode.
	80 GRAPH I1, C, O1	Places graphic area 1 in the input mode, then clears it and places it in the output mode.
SET		Sets a dot in the specified position in a graphic area operating in the input mode. The first operand specifies the X-coordinates (0-319) and the second operand specifies the Y-coordinates (0-199).
	300 SET 160, 100	Displays a dot in the center of the screen.
RESET		Resets a dot in the specified position in a graphic area operating in the input mode.
	310 RESET 160, 100	Resets a dot from the center of the screen.

LINE	400 LINE 110, 50, 210, 50, 210, 150, 110, 150, 110, 50	Draws lines connecting positions specified by operands. Draws a square the length of whose side is 100 in the center of the display screen.										
BLINE		Draws black lines connecting positions specified by operands.										
POSITION	20 GRAPH I2, C, O2 30 POSITION 0, 50 40 PATTERN 8, A\$	Sets the location of the position pointer in a graphic area. The PATTERN statement (see below) is executed starting at the location indicated by the position pointer. Each graphic area has an individual position pointer. Places graphic area 2 in the input mode, sets the position pointer to the position corresponding to the position on the display screen which is at (0, 50), then transfers data from variable A\$ to graphic area 2 so that the pattern corresponding to the contents of A\$ is drawn on the screen starting at (0, 50).										
PATTERN	10 C\$ = "ABCDEF" 20 PATTERN 4, C\$ 30 PATTERN -4, C\$	Draws the dot pattern specified by operands in a graphic area which is in the input mode. Each dot pattern unit consists of 8 dots arranged horizontally and corresponds to 8 bits representing a character. Elements are stacked in the number of layers specified by the value of the first operand and the direction in which layers are stacked is specified by the sign of the first operand. Draws the dot pattern shown as follows.										
		4 layers { 										
		Draws the following dot pattern.										
		4 layers { 										
POINT	100 ON POINT (X, Y) GOTO 10, 20, 30	Ascertaines the dot (X, Y) whether it is set or reset, and branches according to the result. <table border="1"> <thead> <tr> <th>Result of the POINT function</th> <th>Point information</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Points in both graphic areas 1 and 2 are reset.</td> </tr> <tr> <td>1</td> <td>Only point in graphic area 1 is set.</td> </tr> <tr> <td>2</td> <td>Only point in graphic area 2 is set.</td> </tr> <tr> <td>3</td> <td>Points in both graphic areas 1 and 2 are set.</td> </tr> </tbody> </table>	Result of the POINT function	Point information	0	Points in both graphic areas 1 and 2 are reset.	1	Only point in graphic area 1 is set.	2	Only point in graphic area 2 is set.	3	Points in both graphic areas 1 and 2 are set.
Result of the POINT function	Point information											
0	Points in both graphic areas 1 and 2 are reset.											
1	Only point in graphic area 1 is set.											
2	Only point in graphic area 2 is set.											
3	Points in both graphic areas 1 and 2 are set.											
POSH		System variable indicating the X-coordinate (horizontal location) of the position pointer.										
POSV		System variable indicating the Y-coordinate (vertical location) of the position pointer.										

4.1.15 Machine language control statements

LIMIT	100 LIMIT 49151	Limits the area in which BASIC programs can be loaded to the area up to address 49151 (\$BFFF in hexadecimal).
	100 LIMIT A	Limits the area in which BASIC programs can be loaded to the area up to the address indicated by variable A.
	100 LIMIT \$BFFF	Limits the area in which BASIC programs can be loaded to the area up to \$BFFF (hexadecimal). Hexadecimal numbers are indicated by a dollar sign as shown at left.
	300 LIMIT MAX	Set the maximum address of the area in which BASIC programs can be loaded to the maximum address of the memory installed.
	200 LIMIT \$BFFF 210 LOAD FD2 "S-R1"	Loads machine language program (object program) "S-R1" in the machine language link area from the diskette in drive 2 when the loading address of the program is \$C000 or higher.
POKE	120 POKE 49450, 175	Stores 175 in address 49450.
	130 POKE AD, DA	Stores data (between 0 and 255) specified by variable DA into the address indicated by variable AD.
PEEK	150 A=PEEK (49450)	Substitutes data stored in address 49450 into variable A.
	160 B=PEEK (C)	Substitutes the contents of the address indicated by variable C into variable B.
USR	500 USR (49152)	Transfers program control to address 49152. This function is the same as that performed by the CALL instruction, which calls a machine language program. When a RET command is encountered in the machine language program, program control is returned to the BASIC program.
	550 USR (AD)	Calls the program starting at the address specified by variable AD.
	570 USR (\$C000)	Calls the program starting at address \$C000.
	600 WOPEN #8, USR (\$C000)	The statement on line 600 opens a file which is to be written by the machine language program called by USR (\$C000) with logical number 8 assigned. At this stage of program execution the USR function is not executed. The statement on line 610 loads the beginning address of the memory area set with variable A\$ into the DE register of the CPU and its length (max. 255 bytes) into the BC register. This enables the program called by USR (\$C000) to obtain data in A\$. It then executes USR (\$C000).
	610 PRINT #8, A\$ 620 CLOSE #8	

<pre>700 ROPEN #9,USR (\$C100) 710 INPUT #9,B\$ 720 CLOSE #9</pre>	<p>The statement on line number 700 opens a file (which is to be read by the machine language program called by USR(\$C100)) with logical number 9 assigned. The statement on line number 710 executes USR (\$C100). The machine language program called loads string data in the memory area starting at the address indicated by the DE register and loads the length of the data string read in the BC register. It then returns program control to the BASIC program. The BASIC program refers to this memory area as BS.</p>
--	---

4.1.16 Printer control statements

<pre>PRINT/P 10 PRINT/P A,A\$ 20 PRINT/P CHR\$(5)</pre>	<p>Performs the nearly same operation as the PRINT statement on the optional printer.</p> <p>Prints the numeric value of A and the character string of variable A\$ on the line printer.</p> <p>Executes paper home feed. (CHR\$(5) is a control code.)</p>
<pre>IMAGE/P 30 IMAGE/P CHR\$(255), "UU"</pre>	<p>Draws a desired dot pattern (image) specified in the operand on the line printer according to the operating mode (image mode 1 or 2).</p>
<pre>COPY/P 10 COPY/P 1 20 COPY/P 2 30 COPY/P 3 40 COPY/P 4</pre>	<p>Causes the printer to copy the character display.</p> <p>Causes the printer to copy the dot pattern set in graphic area 1.</p> <p>Causes the printer to copy the dot pattern set in graphic area 2.</p> <p>Causes the printer to copy the dot pattern set in both graphic area 1 and graphic area 2.</p>
<pre>PAGE/P 100 PAGE/P 20</pre>	<p>Specifies 20 lines to be contained in one page of the MZ-80P5 line printer.</p>

4.1.17 I/O input/output statements

<pre>INP 10 INP @12,A 20 PRINT A</pre>	<p>Reads data on the specified I/O port.</p> <p>The statement on line number 10 reads data on I/O port 12.</p>
<pre>OUT 30 B = A^2 + 0.3 40 OUT @13,B</pre>	<p>Outputs data to the specified I/O port.</p> <p>The statement on line 40 outputs the value of B to I/O port 13.</p>

4.1.18 Arithmetic functions

ABS	100 A = ABS (X)	Substitutes the absolute value of variable X into variable A. X may be either a constant or an expression. Ex) ABS (-3) = 3 ABS (12) = 12
INT	100 A = INT (X)	Substitutes the greatest integer which is less than X into variable A. X may be either a numeric constant or an expression. Ex) INT (3.87) = 3 INT (0.6) = 0 INT (-3.87) = -4
SGN	100 A = SGN (X)	Substitutes one of the following values into variable A: -1 when X<0, 0 when X=0 and 1 when X>0. X may be either a constant or an expression. Ex) SGN (0.4) = 1 SGN (0) = 0 SGN (-400) = -1
SQR	100 A = SQR (X)	Substitutes the square root of variable X into variable A. X may be either a numeric constant or an expression; however, it must be greater than or equal to 0.
SIN	100 A = SIN (X)	Substitutes the sine of variable X in radians into variable A. X may be either a numeric constant or an expression. The relationship between degrees and radians is as follows. $1 \text{ degree} = \frac{\pi}{180} \text{ radians}$
	110 A = SIN (30 * π /180)	Therefore, when substituting the sine of 30° into A, the statement is written as shown on line number 110 at left.
COS	200 A = COS (X)	Substitutes the cosine of variable X in radians into variable A. X may be either a numeric constant or an expression. The same relationship as shown in the explanation of the SIN function is used to convert degrees into radians. The statement shown on line number 210 substitutes the cosine of 200° into variable A.
	210 A = COS (200 * π /180)	
TAN	300 A = TAN (X)	Substitutes the tangent of variable X in radians into variable A. X may be either a numeric constant or an expression. The statement on line number 310 is used to substitute the tangent of numeric variable Y in degrees into variable A.
	310 A = TAN (Y * π /180)	
ATN	400 X = ATN (A)	Substitutes the arctangent of variable A into variable X in radians. A may be either a numeric constant or an expression. Only the result between - $\pi/2$ and $\pi/2$ is obtained. The statement on line number 410 is used to substitute the arctangent in degrees.
	410 Y = 180/ π * ATN (A)	

EXP	100 A = EXP (X)	Substitutes the value of exponential function e^x into variable A. X may either a numeric constant or an expression.
LOG	100 A = LOG (X)	Substitutes the value of the common logarithm of variable X into variable A. X may be either a numeric constant or an expression; however, it must be positive.
LN	100 A = LN (X)	Substitutes the natural logarithm of variable X into variable A. X may be either a numeric constant or an expression; however, it must be positive.
	110 A = LOG (X)/LOG (Y) 120 A = LN (X)/LN (Y)	To obtain the logarithm of X with the base Y, the statement on line number 110 or line number 120 is used.
RND		This function generates random numbers which take any value between 0.00000001 and 0.99999999, and works in two manners depending upon the value in parentheses.
	100 A = RND (1) 110 B = RND (10)	When the value in parentheses is positive, the function gives the random number following the one previously given in the random number group generated. The value obtained is independent of the value in parentheses.
	100 A = RND (0) 110 B = RND (-3)	When the value in parentheses is less than or equal to 0, the function gives the initial value of the random number group generated. Therefore, statements on line numbers 100 and 110 both give the same value to variables A and B.

4.1.19 String control functions

LEFT \$	10 A\$ = LEFT\$ (X\$, N)	Substitutes the first N characters of string variable X\$ into string variable A\$. N may be either a constant, a variable or an expression.
MID \$	20 B\$ = MID\$ (X\$, M, N)	Substitutes the N characters following the Mth character from the beginning of string variable X\$ into string variable B\$.
RIGHT \$	30 C\$ = RIGHT \$ (X\$, N)	Substitutes the last N characters of string variable X\$ into string variable C\$.
SPACE \$	40 D\$ = SPACE \$ (N)	Substitutes the N spaces into string variable D\$.
STRING \$	50 E\$ = STRING \$ ("*", 10)	Substitutes the ten repetitions of "*" into string variable E\$.
CHR \$	60 F\$ = CHR \$ (A)	Substitutes the character corresponding to the ASCII code in numeric variable A into string variable F\$. A may be either a constant, a variable or an expression.

ASC	70 A = ASC (X\$)	Substitutes the ASCII code (in decimal) corresponding to the first character of string variable X\$ into numeric variable A.
STR\$	80 N\$ = STR\$ (I)	Converts the numeric value of numeric variable I into string of numerals and substitutes it into string variable N\$.
VAL	90 I = VAL (N\$)	Converts string of numerals contained in string variable N\$ into the numeric data as is and substitutes it into numeric variable I.
LEN	100 LX = LEN (X\$)	Substitutes the length (number of bytes) of string variable X\$ into numeric variable LX.
	110 LS = LEN (X\$ + Y\$)	Substitutes the length (number of bytes) of string variable X\$ and Y\$ into numeric variable LX.

4.1.20 Tabulation function

TAB	10 PRINT TAB (X); A	Displays the value of variable A at the Xth position from the left side.
------------	---------------------	--

4.1.21 Arithmetic operators

The number to the left of each operator indicates its operational priority. Any group of operations enclosed in parentheses has first priority.

① ^	10 A = X^Y (power)	Substitutes X^Y into variable A. (If X is negative and Y is not an integer, an error results.)
② -	10 A = -B (negative sign)	Note that “-” in -B is the negative sign and “-” in 0-B represents subtraction.
③ *	10 A = X*Y (multiplication)	Multiplies X by Y and substitutes the result into variable A.
④ /	10 A = X/Y (division)	Divides X by Y and substitutes the result into variable A.
④ +	10 A = X + Y (addition)	Adds X and Y and substitutes the result into variable A.
④ -	10 A = X - Y (subtraction)	Subtracts X from Y and substitutes the result into variable A.

4.1.22 Logical operators

=	10 IF A=X THEN ...	If the value of variable A is equal to X, the statement following THEN is executed.
	20 IF A\$ = "XYZ" THEN ...	If the content of variable A\$ is "XYZ", the statement following THEN is executed.
<> or <	10 IF A <> X THEN ...	If the value of variable A is not equal to X, the statement following THEN is executed.
>= or =>	10 IF A >= X THEN ...	If the value of variable A is greater than or equal to X, the statement following THEN is executed.
<= or =<	10 IF A <= X THEN ...	If the value of variable A is less than or equal to X, the statement following THEN is executed.
*	40 IF (A > X) * (B > Y) THEN ...	If the value of variable A is greater than X and the value of variable B is greater than Y, the statement following THEN is executed.
+	50 IF (A > X) + (B > Y) THEN ...	If the value of variable A is greater than X or the value of variable B is greater than the value of Y, the statement following to THEN is executed.

4.1.23 Other symbols

?	200 ? "A + B="; A + B 210 PRINT "A + B="; A + B	Can be used instead of PRINT. Therefore, the statement on line number 200 is identical in function to that on line number 210.
:	220 A=X : B=X^2 : ? A, B	Separates two statements from each other. This separator is used when multiple statements are written on the same line. Three statements are written on line number 220.
;	230 PRINT "AB"; "CD"; "EF"	Displays characters to the right of separators following characters on the left. The statement on line 230 displays "ABCDEF" on the screen with no spaces between characters.
	240 INPUT "X =" ; X\$	Displays "X=" on the screen and awaits entry of data for X\$ from the keyboard.
'	250 PRINT "AB", "CD", "E"	Displays character strings in a tabulated format; i.e. AB first appears, then CD appears in the position corresponding to the starting position of A plus 10 spaces and E appears in the position corresponding to the starting position of C plus 10 spaces.
	300 DIM A(20), B\$(3, 6)	A comma is used to separate two variables.
" "	320 A\$ = "SHARP BASIC" 330 B\$ = "MZ-80B"	Indicates that characters between double quotation marks form a string constant.
\$	340 C\$ = "ABC" + CHR\$(3)	Indicates that the variable followed by a dollar sign is a string variable.
	500 LIMIT \$BFFF	Indicates that numeric data following a dollar sign is represented in hexadecimal notation.
π	550 S = SIN (X * π / 180)	π represents 3.1415927 (ratio of the circumference of a circle to its diameter).

APPENDIX

The Appendix includes the following;

- *ASCII Code Table Table A.1*

- *DISK BASIC interpreter SB-6510 Error Message Table Table A.2*



This table lists all the possible errors which may occur during program execution. The interpreter notifies the operator of occurrence of an error during program execution or operation in the direct mode with the corresponding error number.

- *Memory Map*

- *Handling diskettes*

A.1 ASCII Code Table

A table of hexadecimal ASCII codes is shown in Figure 2.22 of the Owner's Manual.

CODE CHARACTER	CODE CHARACTER	CODE CHARACTER	CODE CHARACTER	CODE CHARACTER					
0	NULL	26		52	4	78	N	104	h
1	↓	27	TAB	53	5	79	O	105	i
2	↑	28		54	6	80	P	106	j
3	→	29		55	7	81	Q	107	k
4	←	30		56	8	82	R	108	l
5	HOME	31		57	9	83	S	109	m
6	CLR	32		58	:	84	T	110	n
7	DEL	33	!	59	;	85	U	111	o
8	INST	34	"	60	<	86	V	112	p
9	GRPH	35	#	61	=	87	W	113	q
10	SFT LOCK	36	\$	62	>	88	X	114	r
11	BREAK	37	%	63	?	89	Y	115	s
12	RVS	38	&	64	@	90	Z	116	t
13	CR	39	'	65	A	91	[117	u
14	L SCRIPT	40	(66	B	92	\	118	v
15	RVS CANCEL	41)	67	C	93]	119	w
16	F1	42	*	68	D	94	^	120	x
17	F2	43	+	69	E	95	_	121	y
18	F3	44	,	70	F	96	`	122	z
19	F4	45	-	71	G	97	a	123	{
20	F5	46	.	72	H	98	b	124	
21	F6	47	/	73	I	99	c	125	}
22	F7	48	0	74	J	100	d	126	~
23	F8	49	1	75	K	101	e	127	↵
24	F9	50	2	76	L	102	f		
25	F10	51	3	77	M	103	g		

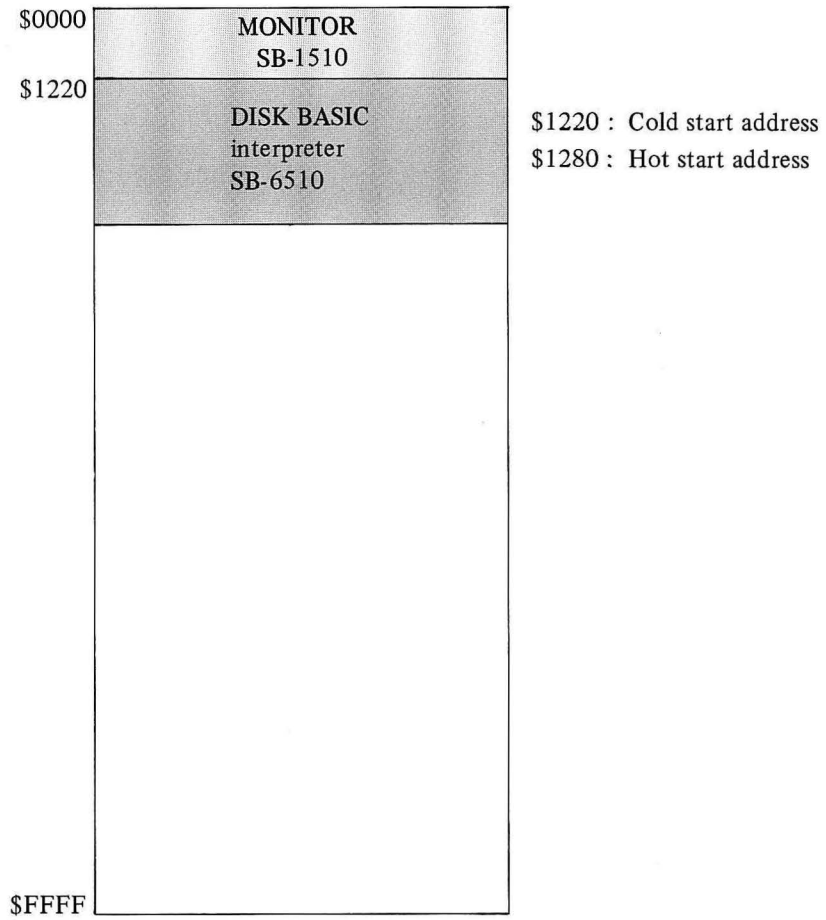
CODE CHARACTER	CODE CHARACTER	CODE CHARACTER	CODE CHARACTER	CODE CHARACTER					
128		154		180	4	206	N	232	h
129		155		181	5	207	O	233	i
130		156		182	6	208	P	234	j
131		157		183	7	209	Q	235	k
132		158		184	8	210	R	236	l
133		159		185	9	211	S	237	m
134		160		186	:	212	T	238	n
135		161	!	187	;	213	U	239	o
136		162	"	188	<	214	V	240	p
137		163	#	189	=	215	W	241	q
138		164	\$	190	>	216	X	242	r
139		165	%	191	?	217	Y	243	s
140		166	&	192	@	218	Z	244	t
141		167	'	193	A	219	[245	u
142		168	(194	B	220	\	246	v
143		169)	195	C	221]	247	w
144		170	*	196	D	222	^	248	x
145		171	+	197	E	223	_	249	y
146		172	,	198	F	224	`	250	z
147		173	-	199	G	225	a	251	{
148		174	.	200	H	226	b	252	
149		175	/	201	I	227	c	253	}
150		176	0	202	J	228	d	254	~
151		177	1	203	K	229	e	255	
152		178	2	204	L	230	f		
153		179	3	205	M	231	g		

A.2 Error Message Table

Error No.	Meaning
1	Syntax error
2	Operation result overflow
3	Illegal data
4	Data type mismatch
5	String length exceeded 255 characters
6	Insufficient memory capacity
7	The size of an array defined was larger than that defined previously.
8	The length of a BASIC text line was too long.
9	
10	The number of levels of GOSUB nests exceeded 16.
11	The number of levels of FOR-NEXT nests exceeded 16.
12	The number of levels of functions exceeded 6.
13	NEXT was used without a corresponding FOR.
14	RETURN was used without a corresponding GOSUB.
15	Undefined function was used.
16	Unused reference line number was specified in a statement.
17	CONT command cannot be executed.
18	A writing statement was issued to the BASIC control area.
19	Direct mode commands and statements are mixed together.
20	RESUME statement cannot be executed.
21	A RESUME statement was used without a corresponding error process.
22	
23	
24	A READ statement was used without a corresponding DATA statement.
25	The number of SWAP levels exceeded 1.
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	

Error No.	Meaning
36	
37	
38	
39	
40	File was not found.
41	Disk drive hardware error.
42	A file name already used was defined again.
43	OPEN, DELETE, RENAME statements were issued to an open file.
44	An unopened file was referenced or a CLOSE or KILL statement was issued to it.
45	
46	A protected file was accessed for writing.
47	
48	
49	
50	The disk drive is not ready.
51	The total number of files on a volume exceeded 94.
52	Volume number error
53	File space on the diskette is insufficient.
54	A diskette which has not been initialized was loaded.
55	The number of data of a BSD file exceeded 64K bytes.
56	Data error occurred on an FDC routine call.
57	The diskette cannot be used.
58	
59	
60	Illegal file name was specified.
61	Illegal file mode was specified.
62	
63	Out of file
64	Illegal logical number was specified.
65	The printer is not ready.
66	Printer hardware error
67	Out of paper
68	
69	
70	Check sum error

A.3 Memory Map



A.4 Handling diskettes

The master diskette must be handled especially carefully. Make a submaster diskette by means of the diskette-copy program in the OBJ file "Utility" on the master diskette. Be sure to keep the master diskette in a safe place.

All optional blank diskettes supplied by the Sharp Co. are not initialized. Be sure to initialize them before use.

Notes on handling of diskettes

- Fingerprints on a diskette may permanently render it unusable. Never touch the diskette surface through the head window.
- Insert the diskette straight into the drive until it stops, then close the front door gently. Rough handling may damage the diskette.
- Do not fold or bend the diskette, or it may be rendered unusable.
- Write the index label before it is affixed to the jacket. If it is written after it is affixed to the jacket, use a felt marker or other soft tip pen.
- Ashes and drinks are the most common contaminants to guard against.
- Ambient temperature: 4~50°C

When the ambient temperature is more than 50°C the jacket may be deformed. Do not place the diskette in a place where it is exposed to direct sun light or where the temperature may exceed 50°C.

Notes on storing diskettes

- Keep the diskettes away from magnets. Even a magnet ring or magnet necklace may damage data on the diskette. Electrical equipment such as the display unit of the computer, a cassette tape recorder, or a TV set generates magnetic flux, so keep diskettes away from such equipment.
- Keep the diskette in the envelope supplied. Make it a habit to put the diskette in the envelope immediately after it has been taken out of the drive. This will prevent almost all problems which result from careless handling of diskettes. The master diskette must be handled especially carefully. The envelopes supplied are made of special materials and guard against static electricity and moisture.
- When storing diskettes for a long time, keep the envelopes in the storage case. Be sure the envelopes are stored vertically in the storage case. Do not incline or bend the envelope. The master diskette is not supplied with a storage case.
- Do not clip diskettes with paper clips or the like.
- Do not place any heavy objects on diskettes.

