# FLEXIBLE DISK UNIT
# TF-20

# HX-20 DISK BASIC REFERENCE MANUAL

# FOREWORD

This manual covers the specifications and use of the hardware and software of the TF-20 Flexible Disk Unit designed exclusively for use with the HX-20 portable computer. To get the fullest use from your TF-20. please read this manual carefully before attempting to operate the unit.

# TABLE OF CONTENTS

## Commands and Statements



## Directions

# 1. INTRODUCTION

## 1.1 Sequential Files and Random Access Files

There are essentially two ways in which data can be accessed (stored and retrieved). These are sequential and random access files.

As the name suggests, sequential files are files in which the data are stored in sequence. Therefore, to retrieve data from the file, the data must be read in sequence and unwanted data must be passed over until the desired data is reached. Among the advantages of sequential files is the fact that, since the data are stored in sequence, the only restriction on the length of a file is the amount of memory available on the storage medium. The simple configuration of sequential files also makes them quite easy to understand and to handle. The main drawback of sequential files is that because unwanted data must be. passed, they are quite slow compared with random access files. The cassette tapes used in the external cassette and the microcassette drive of the HX-20 are both sequential storages.

In a random access file, it is possible to go directly to the location in the storage medium which you wish to read or write. This saves a great deal of time and makes random access files faster to access than sequential files. However, for random access files, in order to be able to locate data easily, it is necessary that all records (the basic unit for handling data in files) be the same length.

In Disk BASIC (the name of the version of BASIC used to operate the TF-20), the fixed length of records is 128 bytes. To fix the length of the variables used in a random access file, the FIELD statement is necessary. LSET and RSET statements are used to store data in the random buffer. The advantage of random access files is that they are very fast and the disadvantages are that they use memory less efficiently than sequential files and must be configured with some care. The TF-20 can be used for both sequential and random access storage.

## 1.2 Flexible Disks

Flexible disks are a lightweight, low-cost storage medium. When the disk is inserted in the disk drive, it is rotated at a speed of the 300 RPM by the spindle in the spindle hole. Access is performed through the read/write access holes by twin read/write heads located on either side of the disk in the drive. The flexible disks used in the TF-20 are double-sided disks which measure 5 1/4" dia. and have an available total storage capacity of 278K bytes. The figure below shows the external view of the disks used in the TF-20.

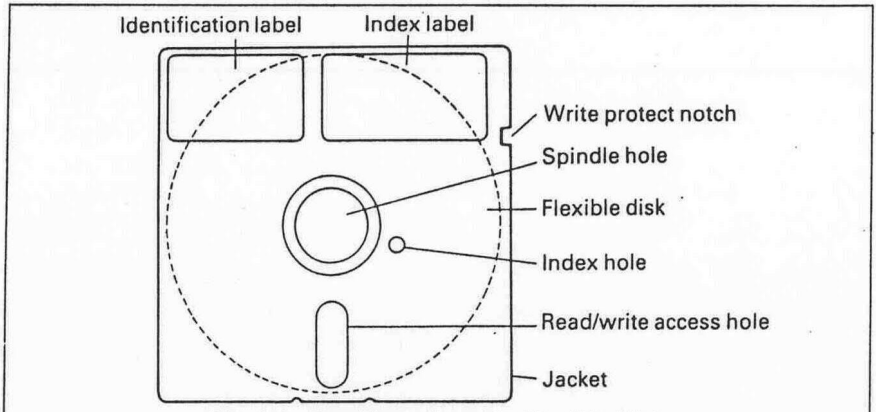



Fig. 1.1 Configuration of a Flexible Disk

Along with the advantages of flexible disks, there are certain conditions which must be met concerning their use. Primarily, these fall under the heading of the care with which flexible disks must be treated. The density with which data is magnetically recorded on a disk, while being the source of its efficiency and economy, is also the reason why care must be exercised in the handling of disks. Careless handling of disks can result in lost or useless data and programs.

Disks are housed ' a protective jacket that serves to clean the recording surface and protect it from dust and other contaminants. The recording surface should never be exposed to oils (including that on your fingers), solvents or a dusty atmosphere. Any of these can destroy the data and programs stored on the disk. The same applies to extreme temperatures or sudden fluctuations in temperature or humidity. Because the disks record data magnetically, disks should never be used or stored near magnets or in a strong magnetic field, such as that produced by a television set. For details of the recording format and complete warnings for use, refer to Chapter 3, Disks.

# 2. GETTING STARTED

## 2.1 Unpacking

When unpacking the TF-20, make sure that all of the following items are included.

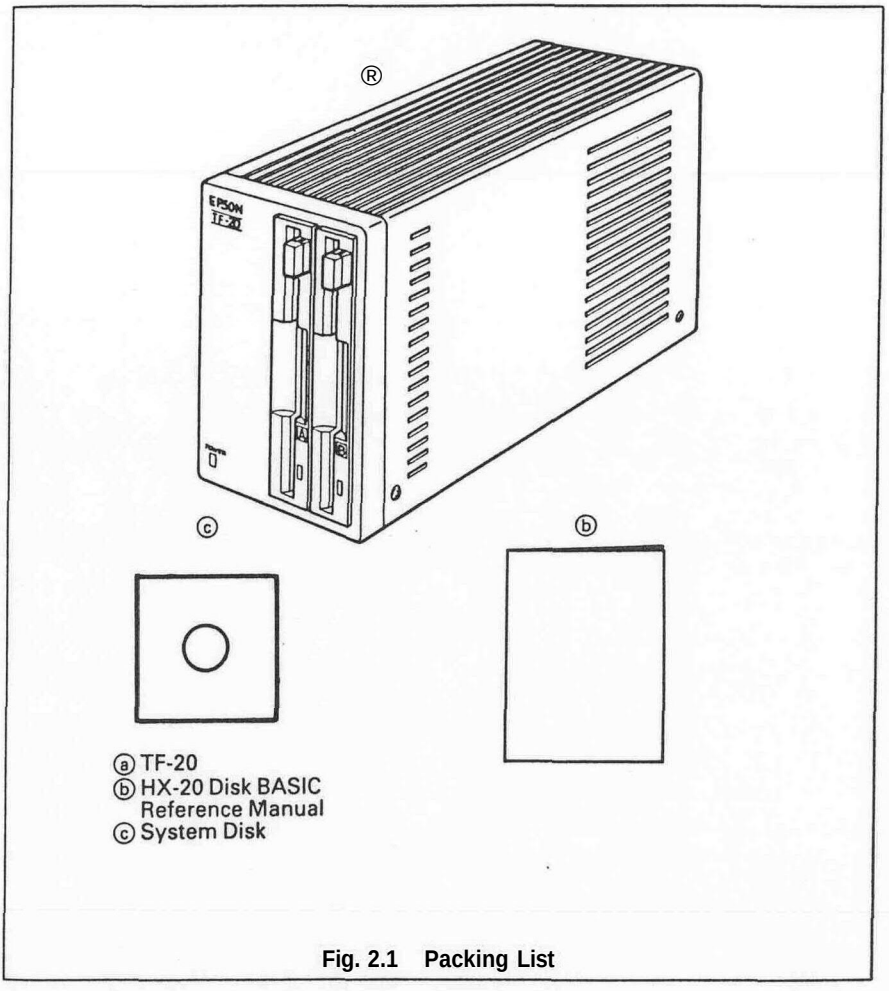
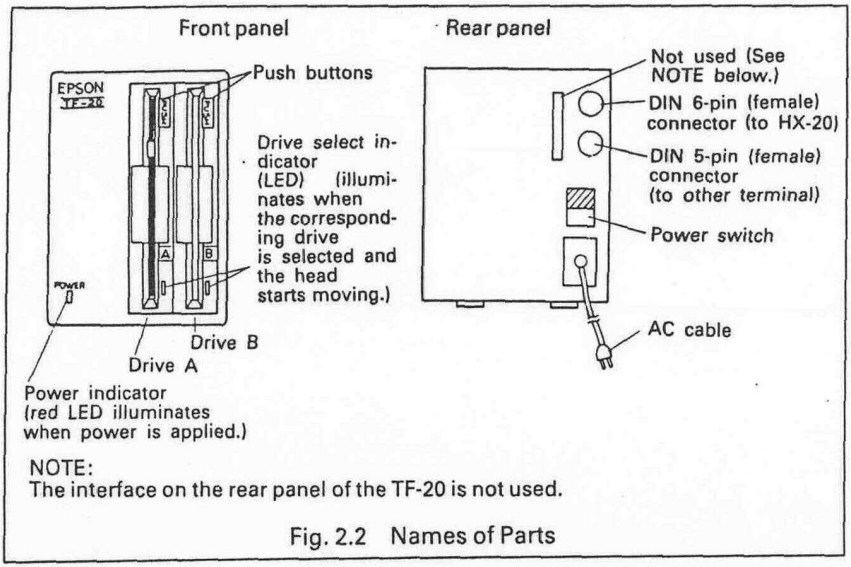



**Fig. 2.1 Packing List**

All packing materials should be saved for reshipment of the TF-20.

## 2.2 Names of Parts




NOTE:
The interface on the rear panel of the TF-20 is not used.

Fig. 2.2 Names of Parts

## 2.3 Hardware Configuration

Up to two TF-20 floppy disk units, each housing two drives, can be connected together in a daisy chain with the HX-20. In Disk BASIC, the designation of the drives of the first unit are drives A and B and those in the second unit, drives C and D. The drive names are determined by the setting of the DIP switches located inside the housing of the TF-20. The factory setting is for drives A and B and the user must reset the DIP switches himself when he adds a second unit. The procedure for adding a second TF-20 is as follows.

(1) Loosen the four screws located on the lower edge of the cover of the second TF-20. Change the setting of DIP switch 4 to OFF. DIP switch 4 is located on the rear, lefthand side of the PC board (in front of the 5-pin interface connector). The location of DIP switch 4 is shown in the Fig. 2.3.

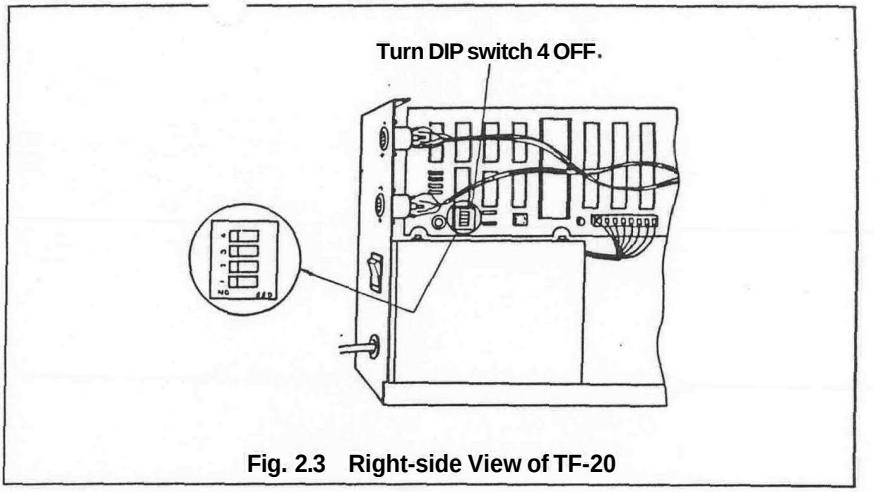



**Fig. 2.3 Right-side View of TF-20**

When DIP switch 4 has been set to the OFF position, the disk drive can be used as the second drive. The device names will be "C:" and "D:".

(2) Connect the DIN 5-pin end of cable #707 to the 5-pin connector of the first TF-20.

(3) Connect the DIN 6-pin end of cable #707 to the 6-pin connector of the second TF-20. "Daisy chaining", as this connection is called, of the HX-20 to two TF-20 drives is shown in the figure below.

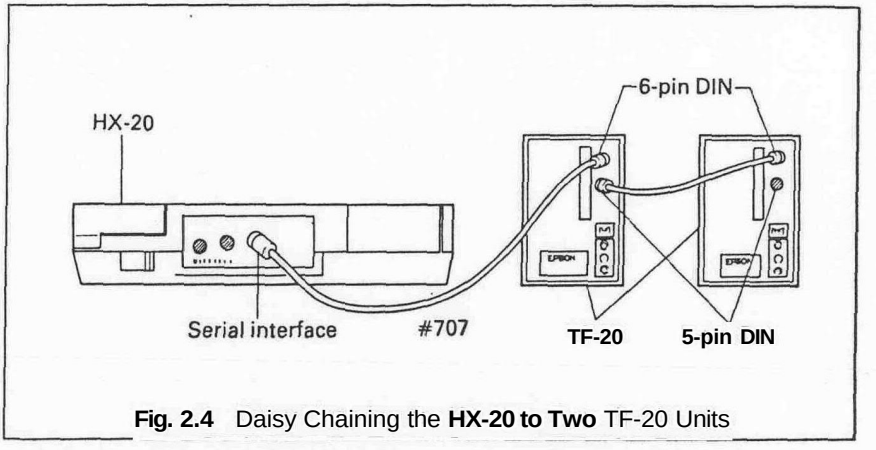



**Fig. 2.4** Daisy Chaining the **HX-20 to Two** TF-20 Units

## 2.4 Operation

### (1) DIP switch setting of the HX-20

Before starting up, DIP switch 4 of the HX-20 must be turned ON If it is not turned ON, you will not be able to boot Disk BASIC even if the TF-20 units are correctly connected to the HX-20.

### (2) Turning the power ON and OFF

The power of the HX-20 must be turned ON after power has been applied to all the terminals· The power of the HX-20 must be turned OFF before turning OFF the power of any of the terminals

Remove any disks set in the TF-20 before turning the power ON or OFF.

### (3) Setting disks in the disk drive

The procedure for setting disks in the disk drive is as follows

STEP 1

Insert the disk into the disk drive so that the write protect notch is at the top of the disk and the identification label is facing right.

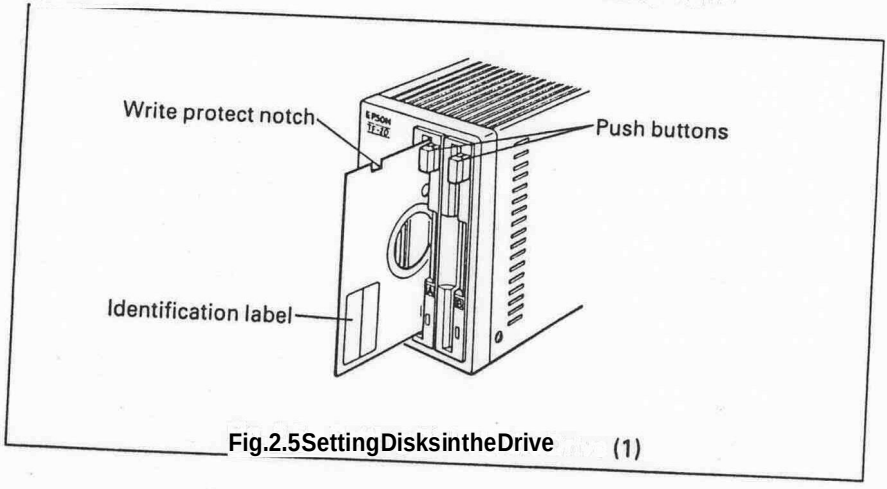



**Fig.2.5SettingDisksintheDrive (1)**

**STEP 2**
**When the disk is inserted, you will hear a sound as the eject lock lever** catches.

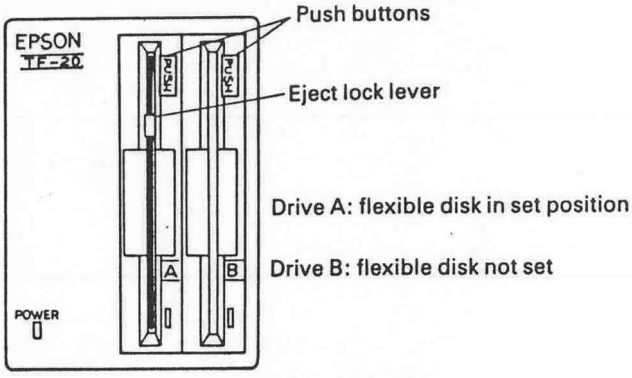



Fig. 2.6 Setting Disks in the Drive (2)

STEP 3
Press the push button slowly and carefully to the ON position. The disk is now set in the drive.

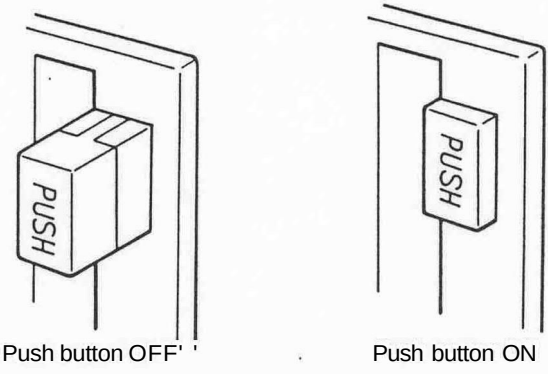



Fig. 2.7 Push Button

### (4) Removing disks from the disk drive

The procedure for removing disks from the disk drive is as follows.

STEP 1

Check to make sure that the drive select indicator is not illuminated before pressing the push button. If the drive select indicator is illuminated, execute CLOSE or RESET statement before you eject the disk. The drive select indicator will turn off a few seconds after the execution of one of the above statements.

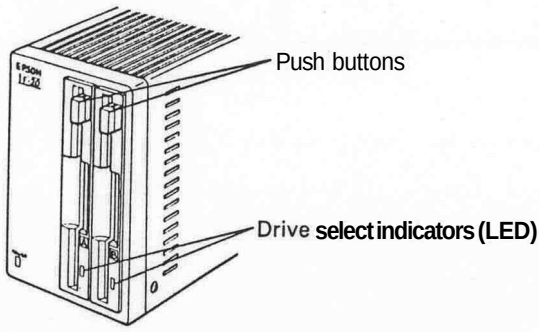



Fig.2.8 Removing Disks from the Drive

STEP 2

Firmly press the push button of the disk drive from which you wish to eject the disk. The push button will return to the OFF position and the disk will be ejected 2-3 cm from the drive.

*CAUTIONS*

1) *Insert the disk into the drive slowly and carefully.*
2) *Check the orientation of the disk (write protect notch up, identification label to the right) before setting the disk in the drive. (If the orientation of the disk is not correct, the TF-20 will not operate.)*
3) *Always remove disks from the drive after checking to make sure that the drive select indicator is extinguished. If you eject a disk while the drive select indicator is illuminated, you risk losing the data stored on your disk.*

# 3. DISKS

## 3.1 Disk Format

The flexible disks used in the TF-20 are double-sided, double-density flexible disks. Double-sided means that data may be stored on both sides (sides 0 and 1) of the disks by means of the read/write heads operating through the access holes. Double density refers to the density with which data can be stored on the disks.

The recording surface of the flexible disk is divided into 40 concentric bands, called tracks. These are referred to, from the outermost to the innermost, as tracks 0 to 39.

Each track is further divided into 16 sectors. These are the basic physical unit of data storage on the disk. At the start of each sector is an area that shows the start of the sector and contains the address data for the sector. This is referred to as the identification or ID field. The ID field is followed by the data field. This is where the data which the user wishes to store is actually written. The amount of data that can be stored in a single sector is 256 bytes, or 4K bytes (1K byte = 1,024 bytes) per track for a total capacity of 320K bytes for the entire disk.

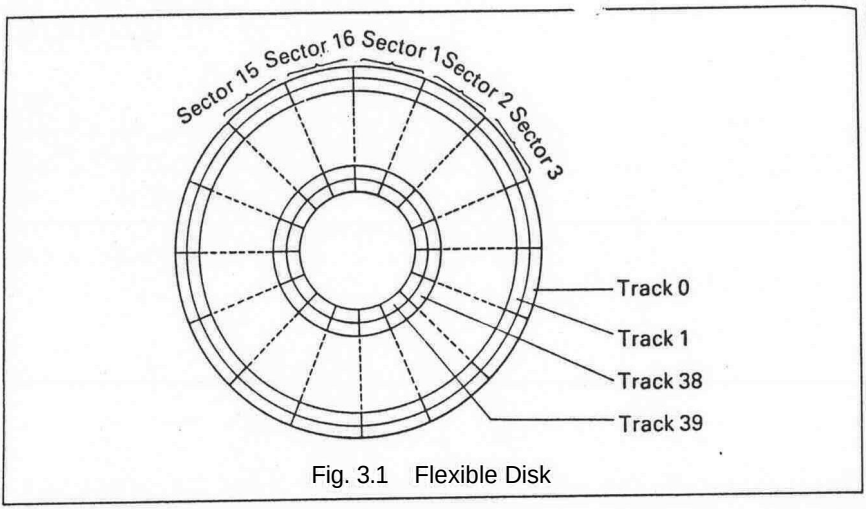



Fig. 3.1 Flexible Disk

Of this total, tracks 0 through 3 (sides 0 and 1) are used as the system area (for the operating system of the TF-20), the first 8 sectors of track 4 (side 0) are used as the directory area and track 39 is reserved. Thus, the file area available for the user is 278K bytes. If the disk is a system disk, the amount will be reduced by the 8K bytes occupied by the system program (which operates Disk BASIC in the HX-20).

For each flexible disk, 64 directory areas are reserved. Thus, the maximum number of files that can be created on a single disk is 64. However, a new directory area is required for every 256 records (32K bytes) of a file and this will have an effect on the total number of files that can be created.

## 3.2 System and Non-System Disks

The disks which can be used in the TF-20 are divided into system and non-system disks. The differences between the two are as follows.

### (1) System disks

A system disk is one which contains the system program for Disk BASIC. To boot Disk BASIC (that is, to load it into the RAM of the HX-20), the system disk must be set in disk drive A. As the system program is written in the user area of the disk, the total amount of data that can be stored on a system disk is somewhat (8K bytes) less than can be stored on a non-system disk.
In the system disk, two files, "BOOT80.SYS" and "DBASIC.SYS" are reserved for the system program. These files are write-protected and will not be displayed by the FILES statement. However, the user should note that he cannot create files using these two filenames.

### (2) Non-system disks

Non-system disks contain only those programs and data which the user has stored on them. Compared with system disks, there is more area available for use by the user.

## 3.3 FORMAT, SYSGEN and COPY

These two commands and one utility program are used to perform necessary operations to your disks: to format them, to create new system disks and to create backup disks.

### (1) FRMAT

Before a disk can be used, the address data must be written into the ID field and all other data necessary for data storage must be present on the disk. The process of writing this data to the disk is called initilization. Disks supplied by EPSON are already initialized and can be used immediately as non-system disks. However, disks other than those supplied by EPSON or those on which data has been destroyed must be initialized by the user. This is done by the FRMAT command.
To format a disk to be usable in the TF-20, input FRMAT followed by the drive name in which the disk you wish to format is set. When a disk is formatted, all of the data on the disk (including the system program) will be destroyed. Therefore, the HX-20 asks for confirmation by displaying the message "Are you sure?". Input "Y" from the keyboard if you wish to execute the statement and "N" if you wish to cancel. See 4.4 Commands and Statements for details.

**(2) SYSGEN**

This command is one of the two methods available for creating new system disks. Since SYSGEN copies only the system area and files whose file type is ".SYS", it does not affect any of the other files on the disk and can be used to convert *a* non-system disk which you are already using into a system disk.

To create a new system disk, place the current system disk in drive A and the disk which you wish to make into the new system disk in drive B. As with the FRMAT command, the HX-20 will ask for confirmation before executing SYSGEN.

For disks that were used as system disks in another system, you must first execute.a FRMAT command before executing SYSGEN. If the disk set in drive B is different from the disk previously set in drive B, you must execute a RESET command before you can execute SYSGEN. See 3.4 Cautions when Replacing Disks and 4.4 Commands and Statements for details.

**(3) COPY**

This is a utility program that can be used to copy files or whole disks. By copying whole disks, it can be used to create new system disks. The entire contents of the disk set in drive A can be copied to the disk in drive B (or similarly, from C to D if two TF-20 disk units are being used). The copy utility can also be used to copy single files as selected by the user. See 4.6 Copy Utility, for details.

## 3.4 Cautions when Replacing Disks

Before changing the disks set in the disk drive, it is imperative that you first execute a CLOSE statement. If the CLOSE statement is not executed, the data in the buffer will not be written as intended to the current disk and may be accidentally written to the next disk inserted in the drive.

Disk BASIC protects disks against this kind of accident by write protecting all newly inserted disks. If an attempt is made to write data to a disk that has been write-protected, a "Disk write protected" error will occur.

The function of the RESET command is to cancel this protection and prepare the disk drive for a new disk. However, it should be noted that the RESET command initializes the disk system and that the contents of any disk open at the time of its execution will be lost. See 4.4 Commands and Statements.

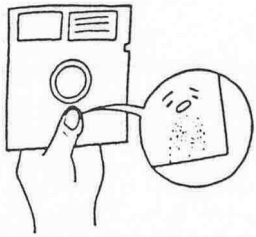
There are also dangers associated with rebooting Disk BASIC after you have left Disk BASIC by such means as turning OFF the power switch of the HX-20 or pressing the MENU key in the middle of program processing.

Rebooting Disk BASIC without turning the power of the TF-20 OFF **or** when an operation in the TF-20 has been halted results in the same conditions as when files have been left open. Therefore, inserting a disk at such a time poses a similar risk of the data on the disk being destroyed. It is a good idea to execute a RESET command after rebooting Disk BASIC under the above mentioned conditions.

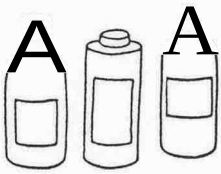
## 3.5 Care and Handling of Flexible Disks

As mentioned earlier, the nature of flexible disks requires that they be handled with a certain amount of care. Please read carefully and observe the below-listed precautions to get the maximum use and service life from your flexible disks.

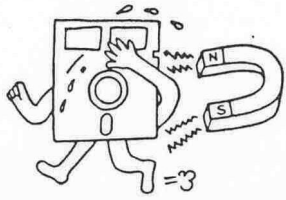
1. Never touch the magnetic recording surface of the disks. The oil from fingerprints on a disk can cause data errors.



2. Never clean the disks with alcohol, freon or other solvents. The data on the disks can be lost if the disk is exposed to the fumes of volatile substances.

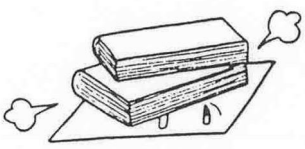



3. Store and use your disks away from magnets and magnetic fields such or those created by a TV. Never hold your disks together with a magnetic paper-weight. Exposure to a magnetic field can cause the magnetically recorded data on the disk to be lost.

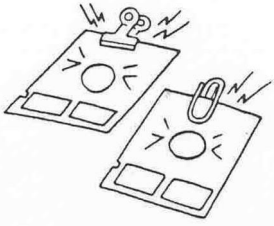



4. Although flexible disks are flexible and can be easily bent, they should never be bent in half or folded. Doing so will render them useless.

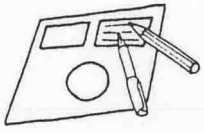
5. Never press the disks or leave them under heavy books, etc. Such treatment will distort the disks and make them useless.



6. Never use paper clips to hold your flexible disks together. This will also distort their shape and be a source of errors.

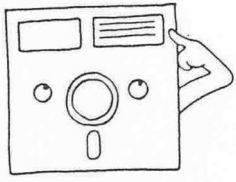


7. Be careful when writing on the identification labels of the disks. Do not use a ball point pen or pencil to write on the labels after they have been affixed to the disk. A felt-tip pen can be used in such a case and will not damage the disk provided that you do not press too hard when writing.

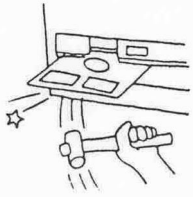


8. Do not use an eraser to erase the identification labels. This creates gummy residue that can damage the disks.

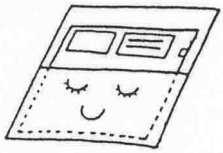
9. Place the identification label correctly in the upper right-hand corner of the disk. Do not affix new labels on top of old ones. The clearances in the disk drive are minimal and this can be a source of error.



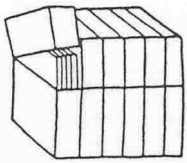
10. Set the disks in the drive slowly and carefully. Forcing the disks or handling them roughly can ruin them for use.



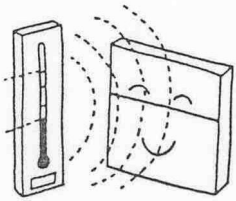
11. Always use the protective envelopes provided with the disks. These envelopes protect the disks from dust and scratches.



12. Store the disks vertically in the storage box. Disks that are left lying about will warp and eventually become unusable.



13. Avoid sudden changes in temperature (more than 20°C within an hour).



## 3.6 Cautions Placement of Disk Drive

The TF-20 is a precision instrument. The following cautions should 2se observed concerning its placement and use.

1. Do not place the disk drive near a heat source or where it will be exposed to direct sunlight.
2. Avoid use or storage of the disk drive where it will be exposed to extreme temperatures or to extreme temperature fluctuations.
3. When using the disk drive continuously for a long period of time, pay careful attention to the ambient temperature.
4. To protect the disk drive against overheating, it has been provided with ventilation holes. Always use the disk drive in a well-ventilated environment and be sure that nothing is placed on top of the drive and that the ventilation holes are not blocked.

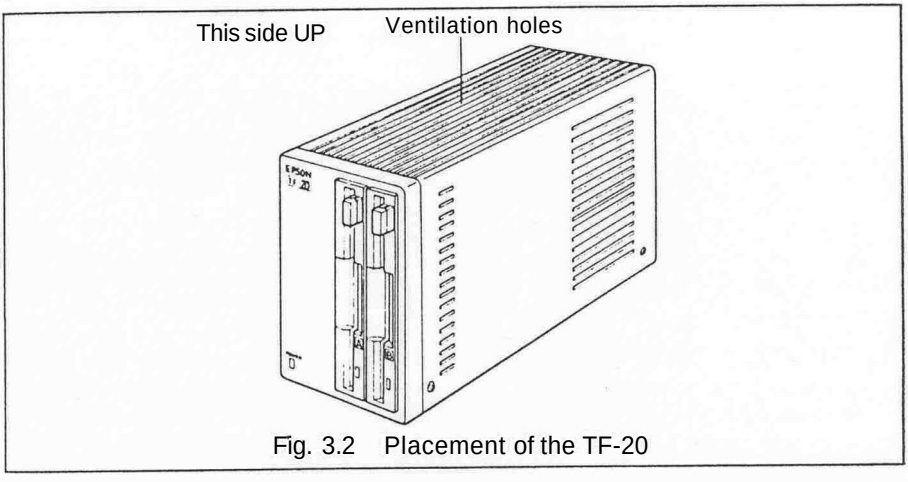



Fig. 3.2 Placement of the TF-20

5. Avoiduseofthediskdriveinaveryhumidatmosphereorwhereitwillbeexposedtooilyormetalli be exposed to oily or metallic dust.
6. The TF-20 is composed of precision parts. Place it where it will be protected from shock and excessive vibration.
7. Place the disk drive as shown in Fig 3 2 Do not place it on its side. Position it as near the horizontal as possible.
8. The TF-20 may malfunction if it is placed near machinery that generates a strong magnetic field. Do not expose the disk drive to magnets or magnetic fields.

9. Do not use an AC outlet which is being used for equipment (such as large motors) that generates electrical noise.
10. Use a commercial AC 100V power source. If the voltage is too high or too low, it can be a source of malfunction.
11. When transporting the TF-20, always insert the transport protection sheets in the drives to protect the read/write heads. (In an emergency, flexible disks can be used for this purpose, although there is a strong risk that the disks will be damaged.) Always transport the disk drive in the packing in which it came.

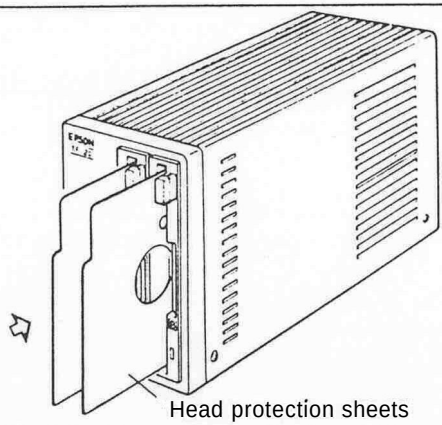



Fig. 3.3 Transporting the Disk Drive

# 4. DISK BASIC

Disk BASIC is an expansion of the ROM BASIC which is normally used to run the HX-20. The interpreter for this expansion is supplied as the system program on the system disk.

When the TF-20 is connected to the HX-20, starting up BASIC will cause the expansion interpreter to be loaded from the disk into the RAM of the HX-20. Then, until the HX-20 returns to the menu, the expansion interpreter will function along with the interpreter already in the ROM to effect the necessary processing for I/O to the disks and for the newly added commands, statements and functions of Disk BASIC. The processing of the previously existing functions continues to be handled by the interpreter in the ROM.

## 4.1 Disk BASIC Features

All of the commands, statements and functions of ROM BASIC are still available for use in Disk BASIC. In addition. Disk BASIC adds the following features.

### (1) I/O to disks

The device names for the first TF-20 connected are "A:" and "B:", and those for a second TF-20, if connected, are "C." and "D:". These device names can be used for any of the ROM version commands and statements related to I/O. The only exception is the LOAD? command.

### (2) Protect SAVE

When the P option is specified in a SAVE statement, the program saved will be protected from listing and saving.

### (3) Features added to ROM BASIC

| Commands and Statements | Functions |
|---|---|
| *DSKO$ | CVI, CVD. CVS |
| FIELD | *DSKF |
| *FILNUM | *DSKI$ |
| *FRMAT | LOC |
| GET | LOF |
| KILL | MKI$. MKD$. MKS$ |
| LSET, RSET | |
| NAME | |
| PUT | |
| *RESET | |
| *SYSGEN | |
| WHILE...WEND | |

***Features unique to HX-20 Disk** BASIC.

## 4.2 Booting Disk BASIC

### (1) Hardware checklist

Before attempting to boot Disk BASIC, make sure that the following hardware conditions have been established.

- DIP switch 4 of the HX-20 is ON.
- Daisy chain cable #707 is correctly connected to the TF-20 and the HX-20.
- Power is applied to all peripheral equipment before the power of the HX-20 is turned ON.

After making all of the above checks, carefully set the system disk in drive A. When you do so, the indicator lamp on the TF-20 will turn on and off and DOS will be loaded into the RAM of the TF-20. Next, start up BASIC in the HX-20. The indicator lamp on the disk drive will again turn on and off and Disk BASIC will be booted in the HX-20. When Disk BASIC has been successfully booted, the display of the HX-20 will look like this.

```
DISK BASIC V-1.0
Copyright 1982 by
Microsoft & EPSON
P1:      0      Bytes
>
```

(2) Problems in booting

If, instead of Disk BASIC, you find that you have activated the ROM BASIC of the HX-20 (that is, if the display is the same as is always displayed when you start up the HX-20), or if the message "CANNOT LOAD" is displayed, the source of the trouble could be any one of the following.

(a) A non-system disk has been set in drive A.
(b) The HX-20 and the TF-20 are not properly connected.
(c) There is a malfunction in the TF-20.

If the message "OUT OF MEMORY" is displayed and ROM BASIC is initialized, this means that there are too many BASIC programs and/or RAM files stored in the RAM and there is no area left to load Disk BASIC.

*NOTE:*
*To start up BASIC in the HX-20 in order to boot Disk BASIC, any of the methods normally used to start up BASIC is fine. You can press the [2] key to select BASIC from the menu, or you can select one of the BASIC programs from the menu. You can even use the Monitor K command to enter BASIC directly when power is applied. Disk BASIC can be loaded correctly regardless of the method used to start BASIC.*

## 4.3 DiskBAS' Syntax

**File descriptor**

File descriptors for disks are expressed in the following format.

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

(a) Device name
In Disk BASIC, in addition to the device names that could be used in ROM BASIC, you can now use device names "A:", "B:", "C:" and "D:" to specify the disk.

| | | |
|---|---|---|
| TF-20 (1) | Left-hand drive | "A:" (default device) |
| | Right-hand drive | "B:" |
| TF-20 (2) | Left-hand drive | "C:" |
| | Right-hand drive | "D:" |

In Disk BASIC, the default device is always drive A. The only exceptions are FRMAT and LOAD?. The default device for FRMAT is "B:" and the function of the LOAD? command remains unchanged from ROM BASIC; either "CAS0:" or "CAS1:" will be selected.

(b) Filename
Filenames in Disk BASIC are expressed in the following format.

<filename> [.<file type>]

As in ROM BASIC, <filename> is a string with a maximum length of 8 characters composed of any of the characters with character codes in the range 1 to 254 with the exceptions of period [ ], colon [ ], parentheses [()] and question mark [?].
<file type> when used for any device other than the disk is a string of up to three characters composed of the same character set that can be used in the specification of the filename. However, if the specified device is one of the disk drives, character codes in which the value of the MSB is 1 cannot be used in the <file type> If such a character is used, the MSB will be changed to 0.

## 4.4 Commands and Statements

The commands and statements used in **Disk BASIC are as follows.**

# CLOSE

**Function** To close file(s).

**Format** CLOSE [[#]<file **number>,[#]<file number>...]]**

**Example** CLOSE #3

**Remarks** CLOSE closes a file specified by <file number>. <file number> is the number under which the file was opened. The file may then be reopened using the same or a different file number; likewise, that file number may now be reused to open any file.

A CLOSE statement without <file number> closes all files opened at the time of executing the statement. # before <file number> may be omitted .Note that END and NEW statements always close all files automatically but a STOP statement does not close any files.

When an output file has been opened, the file must be closed in order to correctly complete the output processing of the data remaining in the buffer. After CLEAR, LOGIN, NEW, DELETE, WIDTH, LOAD. RUN, or MERGE is executed, or a program is edited, all the files being open at that time will be closed.

(See OPEN.)

# DSKO$

**Function** To write data directly to a disk.

**Format** **DSKO$ <drive name>, <track No.>, <sector No.>, <string expression>**

**Example** **DSKO$ "A:", 1, 5, A$**

**Remarks** This command is a disk output command to write the contents of <string expression> to a specified location on the disk in the drive indicated by <drive name>. Data write is in units of one record (128 bytes) and is physically written to the first or second half (128 bytes) of a single sector. The length for <string expression> is 128 characters. If a string longer than this length is used, the first 128 bytes **of** data will be accepted and the remainder will be ignored. If fewer than 128 characters are input, an FC error will occur. To store data which is less than 128 bytes, the remaining bytes should be padded with dummy data (OOH) as is done in the following program.

```
5 CLEAR 500
10 A$="12345fc"
20 ft$=LEFT$(A$+STRING$(1
27,CHR$(0)),128)
re DSKO$ "A:",8,18-A$
```

<drive name> can be "A:", "B:", "C:" or "D:".
<track No.> can be any integer from 0 to 39.
<sector No.> can be any integer from 1 to 64.

*NOTE:*
*In this command, <sector No.> refers to a logical sector (the amount of data transferred in a single logical I/O operation) or one record (128 bytes). As the length of the physical sectors is 256 bytes, <sector No.> 1 indicates the first half of the first physical sector, <sector No.> 2 indicates the second half of the first physical sector, etc. This pattern is continued! in the following manner.*

| Physical Sector | | | <sector No.> |
|---|---|---|---|
| SideO | 1 | First half | 1 |
| | | Second half | 2 |
| | 2 | First half | 3 |
| | | Second half | 4 |
| | | . | |
| | | . | |
| | | . | |
| | 16 | First half | 31 |
| | | Second half | 32 |
| Sidel | 1 | First half | 33 |
| | | Second half | 34 |
| | | . | |
| | | . | |
| | | . | |
| | 16 | First half | 63 |
| | | Second half | 64 |

(See DSKI$.)

# FIELD

**Function** To define the field of the random file buffer.

**Format** **FIELD [#] <file No.>, <field length> AS <string variablo [,<field length> AS <stnng variable>...]**

**Example** **FIELD #1, 20 AS A$, 35 AS B$**

**Remarks** Before data can be written into (PUT) or read from (GET) the random file buffer, the FIELD statement must be executed. <file No > is the number specified by the OPEN statement when the file was opened and <field length> is the number of characters assigned to <string variable>. The total number of characters assigned as <field length> by the FIELD statement cannot exceed 128. If it does, a "Field overflow" error will occur.

(See OPEN, GET, PUT, LSET/RSET.)

# INPUT#

**Function** To read a single item of data from a sequential file and assign it to a variable.

**Format** **INPUT# <file No.>, <variable list>**

**Example** **INPUT# 1, A, B$**

**Remarks** <file No.> is the number used when the file was opened for input by the OPEN statement. <variable list> lists the variables to which the data is to be assigned. However, the type of the variables must match that of the data which is assigned to them. The format of the data items in the file will vary depending on the input device. For devices other than the disk drive, see BASIC Reference Manual.

When numeric data is input, space, CR (ODH), LF (OAH), etc., at the beginning of the line is ignored and any other initial character is taken as the start or numeric data. Numeric data is delimited by spaces, CR (ODH) or commas [, ] When string data is input, space, CR (ODH), LF (OAH), etc., at the beginning of the line is ignored and any other initial character is taken as the start of string data. When the initial character is a quotation mark [" ], all data until the next quotation mark will be read as a single item of string data. When the initial character of the string is not a quotation mark, that string is delimited by commas or CR. If the end of the file (EOF) is reached before the start of numeric or string data read, an IE will occur. If EOF is reached in the middle of data read, that data item will be delimited at that point. Also, if an attempt is made to read invalid characters into a numeric variable, or invalid delimiters are used, a BD error will occur.

(See PRINT#, LINE INPUT# and OPEN.)

# KILL

**Function** To delete a file from the disk.

**Format** **KILL <file descriptor>**

**Example** **KILL "B:TEST.BAS"**

**Remarks** <file descriptor> must indicate a disk file. A KILL statement should not *be* executed for a currently open file. If it is, the results cannot *be* guaranteed. A KILL statement is effective for all types of disk files (BASIC program, BASIC data and machine language program files).

# LINE INPUT#

**Function** To read an entire line from a sequential data file to a string variable.

**Format** **LINE INPUT#<file number>, <string variable>**

**Example** **LINE INPUT#1, A$**

**Remarks** LINE INPUT# inputs an entire line of characters (255 characters max.) up to a carriage return from a sequential file without the use of delimiters and assigns it to <string variable>.
<file number> is the number under which the file was opened by an OPEN statement. <string variable> is the variable name to which the line will be assigned.
LINE INPUT# is especially useful if each line of a data file has been broken into fields, or if a BASIC program saved in ASCII mode is being read as data by another program.

(See INPUT#)

# LIST

**Function** To output a program list to the specified file.

**Format** **LIST <file descriptors [,[<line number>] [-[<line number>]]]**

**Example** LIST "B:TEST.ASC", 3080

**Remarks** The function of LIST when followed by a file descriptor is the same as ASCII format SAVE. When a string or string variable is used to specify the file, it must always be enclosed by double quotation marks.

# LOAD

**Function** To load a program file into the memory.

**Format** **LOAD [<file descriptor>[,R]]**

**Example** **LOAD "B:PROG1.ASC"**

**Remarks** This command loads the program file specified by <file descriptor> into the memory. When LOAD is executed, all open files are closed and all current variables are deleted. However, if the <R> option (i.e., load and run) is used with LOAD, all open files are left open and the program is run immediately after it is loaded.

If <file descriptor> is omitted, the first file of the default device is loaded. LOAD retains the programs currently residing in the memory until the specified file is found and actual loading begins.

Before executing LOAD, use the STAT command to check the area currently logged in.

Attempting to LOAD in an area which has been named by a TITLE statement, will result in the occurrence of a PP ("Protected program") error.

(See SAVE.)

# LOADM

**Function** To load a machine language program file into the memory.

**Format** **LOADM [<file descriptor>][,[offset value][,R]]**

**Example** **LOADM "B:ABC"**

**Remarks** The file to be loaded should be a machine language program file created by the monitor function or the SAVEM command. If <file descriptor> is omitted, the first file of the default device is loaded.

<offset value> is added to the top address specified by the SAVEM command and loading begins at the resulting address.

If the <R> option is specified, after the machine language subroutine is loaded into the memory, program execution begins at <execution starting address> specified by the SAVEM command. If <R> is not specified, the HX-20 returns to BASIC command level after the machine language program has been loaded.

However, with a SAVEM command, the contents of the memory can be created as a file. If <R> is used with a LOADM command in a case other than machine language program file, the CPU will interpret this file as a machine language program file and will execute loading accordingly. If this happens, the BASIC programs and RAM files may be destroyed. Please be careful when specifying <R> option.

<offset value> cannot be specified when the machine language subroutine is not relocatable, i.e., if it cannot be executed when it is loaded at a location different from that at which it was saved. However, as the CPU performs no check to determine whether <offset value> is permitted or not there may be cases in which a file is moved to a different address by the <offset value> even though the file contains the memory data.

With LOADM, "COM0:" cannot be specified as a device name.

(See SAVEM.)

# FILES

**Function** To display the filenames in the specified device.

**Format** **FILES [<file descriptor>]**

**Example** **FILES "B: * . BAS"**

**Remarks** When the device specified in <file descriptor> is not the TF-20, (i.e., "CAS0:", "CAS1:" or "PAC0:"), the FILES statement displays the names of the files in the specified device in the format described in the BASIC Reference Manual.

When the device specified by <file descriptor> is the disk unit ("A:", "B:", "C:" or "D:"), only those files which conform to the <filename> and <file type> specified in the <file descriptor> will be displayed. If only the device is specified, all of the files on that disk will be displayed in the following format:

filename, file type

At this time, in addition to the characters that can normally be used to specify the range of the file, question mark [?] and asterisk [*] can also be used within the <file descriptor>. Any character will be accepted in the position which is occupied by the question mark, and any <filename> or <file type> will be accepted as conforming to a <filename> or <file type> which begins with an asterisk.

# FILNUM

**Function** To specify the number of file control blocks (FCB) that can be used in a disk file.

**Format** **FILNUM <No. of FCB>**

**Example** **FILNUM 5**

**Remarks** FILNUM specifies the number of files that can be opened simultaneously during the execution of a BASIC program. When a disk file is opened, for each file, a 143-byte file control block (FCB) is necessary. FILNUM specifies in advance the size of the area to be reserved for the FCBs. <No. of FCB> can be a value from 1 to 15. When FILNUM is executed, all files open at that time are closed. Also, data specified by the DEFINT and ON ERROR statements become invalid.

(See OPEN and CLOSE.)

# FRMAT

**Function** To format the disk.

**Format** FRMAT [<drive **name>]**

**Example** FRMAT "A:"

**Remarks** The FRMAT statement formats the disk in the specified disk drive.

<drive name> is a string with a value of "A:" through "D:". If omitted, drive "B:" is assumed. The computer will print the statement "Are you sure?" to ask for confirmation. Input "Y" from the keyboard if you wish to execute the statement and "N" if you wish to cancel.

When using a non-EPSON disk for the first time, formatting is necessary before use. It is also necessary when you wish to reuse a disk on which data has been destroyed and which can no longer be read.

At the same time that the disk is formatted, the directories are also initialized so that it can be used immediately as a non-system disk. Also, it must be kept in mind that the FRMAT statement destroys all the data currently on the disk.

# GET

**Function** To read one record from a random disk file into the random buffer.

**Format** **GET *[#]* <file No.> [,<record No.>]**

**Example** **GET #1, 3**

**Remarks** <file number> is the number under which the file was opened by the OPEN statement If <record No.> is specified, that record will be read. If it is omitted, the record immediately following the record used in the most recent GET or PUT statement will be read <record No > must be in the range of 1 to 32767. If it is not, a "Bad record number". "0V" or "FC" error will occur. Also, if a <record No.> which exceeds the size of the current file, or is in a block whose area has not been reserved, is specified, a dummy record (data 00H) will be read by the GET statement. The file specified in the GET statement must have been opened as a random file.

(See FIELD, OPEN, PUT.)

# MERGE

**Function** To merge a specified program file into the program currently in memory.

**Format** **MERGE [<file descriptor>[,R]]**

**Example** **MERGE "A:PROG3.ASC"**

**Remarks** MERGE command merges the program file specified by <file descriptor> into the program in the memory area currently logged in. The specified file must have been saved in ASCII format. If not, a BF ("Bad file mode") error occurs. If <file descriptor is omitted, the first file of the default device will be read. If any lines in the file have the same line numbers as lines in the program in the memory, the lines in the file will replace the corresponding lines in the memory. If the option R is specified, the merged program will be executed after the MERGE operation. BASIC always returns to command level after executing a MERGE command. When a MERGE command is executed, all files open at that time are closed and all variables are cleared. However, if <R> is specified, MERGE is executed with the files being left open.

(See SAVE)

# LSET, RSET

**Function** To store left- or right-justified data in the random file buffer.

**Format**
**LSET <string variable> = <string>**
**RSET <string variable> = <string>**

**Examples**
**LSET A$ = "EPSON"**
**RSET B$ = "TF-20"**

**Remarks** To store data in the random file buffer, RSET or LSET must be used.

When the length of <string> is shorter than the length assigned to <string variable> in the FIELD statement, the string will be left- or right-justified (padded with spaces) to fill the assigned length for the variable.

If the length of <string> is greater than that assigned in the FIELD statement, the excess length on the right end of the string will be lost whether the statement executed is LSET or RSET. Before numeric values can be stored in the random file buffer, they must first be type-converted to strings by the MKI$, MKS$ or MKD$ functions. LSET and RSET can also be used for strings which have not been allocated to the random file buffer by the FIELD statement.

# NAME

**Function** To change the name of a disk file.

**Format** **NAME <current file descriptor> AS <new file descriptor**

**Example** **NAME "TEST" AS "TEST 1"**

**Remarks** The file specified in <current file descriptor> must exist in the specified disk and the file specified in <new file descriptor> must not already exist in the specified disk. If these conditions are not met, an NE or "File already exists" error will occur. Also, the device specified in both the current and the new file descriptor must be the same. NAME statements should not be executed against files which are currently open. The results of such an execution cannot be guaranteed.

# OPEN

**Function** To open a file for I/O.

**Format** **OPEN <mode>, *[#]* <file No.>, <file descriptor>**

**Example** **OPEN "R", # 1, "B:TEST.DAT"**

**Remarks** This statement opens the file specified by <file descriptor> under the specified <file No.>. The OPEN statement allocates buffer area for I/O. For disks, a single FCB is assigned for each file, regardless of mode. Therefore, as many files as were specified in the FILNUM statement can be opened simultaneously. <mode> may be "0", "I" or "R".

0 . . . sequential output mode
I . . . sequential input mode
R . . . random I/O mode

R can only be used for a disk file. When a disk file is opened in either the 0 or R mode, if the file does not exist on the specified file, a new file will be created. If the file already exists, that file will be used, in which case the next PRINT# statement in the 0 mode will output data to the file starting at its beginning and all the data in the file will be lost. When a disk file is opened in the I mode, if the specified file does not exist, an NE error will occur.
More than one OPEN statement, each with a different <file No.> can be used at the same time in relation to a single file. However, when output is performed to the file, it cannot be open under any other <file No.>. The results of outputting to a single file using multiple file numbers cannot be guaramteed.

(See CLOSE and FILNUM.)

# PRINT#

**Function** To output data to a sequential file.

**Format** PRINT# **<file No.> [,<expression>]**

**Example** PRINT# 1, A$

**Remarks** <file No.> is the number specified when the file was opened for sequential output.
<expression> is the string or numeric data to be output to the file. The format in which the data will be output varies depending on the device used for output. For the output format for devices other than the disk drive, refer to the BASIC Reference Manual.
The format output to the file is exactly the same as that output to the display with the PRINT statement. However, as the length of the output line is unlimited, CR or LF are not output automatically.

(See INPUT#.)

# PRINT# USING

**Function** To write strings and numerics into a sequential file using a specified format.

**Format** **PRINT#<file number>, USING<"format string">; [<expression>[ ; <expression>. . .]]**

**Example** **PRINT#1, USING "###";A**

**Remarks** PRINT# USING writes string or numeric expressions into the sequential file specified by <file number> using the format specified by <"format string">. For <"format string">, refer to PRINT USING.

PRINT# USING outputs data in almost the same format as that for output to the display screen. Therefore, when reading a data file output by a PRINT USING, the data will not be delimited unless you use delimiters, commas for numeric expressions and double quotation marks for string expressions.

In general, if the data does not end with a colon or semicolon, carriage return and line feed will be output. However, if the output device is "CAS0:" or "CAS1:", only carriage return will be output.

(See OPEN and PRINT#.)

# PUT

**Function**

To write one record from the random file buffer into the random disk file.

**Format**

**PUT [#] <file No.> [,<record No.>]**

**Example**

**PUT# 1, 3**

**Remarks**

<file No.> is the number specified in the OPEN statement when the file was opened.

If <record No.> is specified, the data will be written into that record. If it is omitted, the record immediately following the record used in the most recent GET or PUT statement will be written to. <record No.> must be in the range of 1 to 32767. If it is not, a "Bad record number", "0V" or "FC" error will occur.

The file against which the PUT statement is executed must have been opened in the random mode.

(See GET. FIELD and OPEN)

# RESET

**Function** To enable the replacement of a disk.

**Format** **RESET [<drive name>]**

**Example** **RESET "C:"**

**Remarks** The disk system (OS) of the TF-20 protects disks against data loss caused by careless replacement procedures by automatically write protecting disks when they are inserted. Any attempt to write on such a disk will result in a "Disk write protected" error. To reset the system and cancel this protection, a RESET statement must be executed after a disk has been replaced. <drive name> can be "A:", "B:", "C:" or "D:". ("C:" and "D:" can only be specified if a second TF-20 is connected.)

If "A:" or "B:" are specified, the first TF-20 is reset and the disks set in either drive can be replaced. In the same way, if "C:" or "D:" is specified, the second TF-20 will be reset, allowing you to replace either of the disks set in that unit. If <drive name> is omitted, "A:" is assumed.

Removing a disk from the drive while there are files still open for output is not recommended as you risk losing the data on the disk. Also, as the RESET statement initializes the system, the contents of any files open when RESET is executed will be lost. Therefore, always execute a CLOSE statement before a RESET to make sure that all open files are closed.

When executing a RESET statement for "A:" and "B:", a disk must be set in drive A. In the same way, when executing a RESET statement for "C:" and "D:", a disk must be set in drive B. If there is not, a DU error will occur.

Sample Programs:

(Direct mode)

```
CLOSE
(replace the disk)
RESET
```

(Program mode)

```
100 CLOSE
110 PRINT "REPLACE DISK
AND PRESS ANY KEY"
110 H$=INPUT$(I)
130 RESET
```

# RUN

**Function** To start program execution.

**Format** **RUN <file descriptor>[,R]**

**Example** **RUN"B:TEST"**

**Remarks** When this command is input, the program specified by <file descriptor> is loaded into the memory and then program execution begins.

For a disk file, if the device name is omitted, "A:" is assumed.

Execution of a RUN command clears all variables and closes all open files before loading the designated program. However, if the <R> option is used with this command, all data files remain OPEN.

Before executing a RUN command, use a STAT command to check the program area currently logged in.

# SAVE

**Function** To save programs to a disk or cassette.

**Format** **SAVE <file descriptor> [ ,A | ,V | ,P ]** ASCII / Microcassette / Binar.

**Example** **SAVE "B:PRG",A**

**Remarks** If the A option is specified, BASIC will save the program in ASCII format. Otherwise, the program will be saved in compressed binary format. ASCII format requires more time and memory but is required when inputting files with the MERGE command. The A option is also used when the program saved is to be used as data.

The <V> option may be used when a microcassette recorder is being used as an auxiliary memory. In this case, after the SAVE is executed, the tape will automatically be rewound to the beginning of the program and program verification (CRC check) will be performed. If <device> other than the microcassette recorder is specified, the <V> option will be ignored.

If the P option is specified, Disk BASIC will save the program in coded binary format. When a program that has been protected in this way is loaded, any attempt to execute a LIST or SAVE command, or to edit the program, will result in a PP error. The P option is effective for devices other than the disk. However, if a protect saved program is loaded when Disk BASIC has not been booted, it will be loaded in code and execution will not be possible.

(See LOAD)

# SAVEM

**Function** To save the memory contents on a specified file.

**Format** **SAVEM <file descriptors <top address>, <bottom address>, <execution starting address>** [,V]

**Example** SAVEM "B:ABC", &H0B00, &H0C00, &H0B00

**Remarks** SAVEM saves a machine language program or memory contents on a specified file.
<top address> and <bottom address> indicate the range of the memory contents to be saved on the specified file.
If the <V> option is specified, program verification (CRC check) is performed after SAVEM is executed. If a device other than the microcassette recorder is used, the <V> option will be ignored.
Execution of machine language program loaded into the memory by a LOADM command will begin at <execution starting address>. Even if the data saved is not a machine language program <execution starting address> cannot be omitted and the same value as the <top address> must be set.

(See LOADM.)

# WHILE . . . WEND

**Function** To conditionally execute the statements in a loop.

**Format**

**WHILE <expression>**

.

**[<loop statements>]**

.

WEND

**Example**

```
WHILE  I<5
  .
  .
  .
WEND
```

**Remarks** If the condition in <expression> is true, the <statements> will be executed until the WEND is encountered. Then, BASIC returns to the WHILE statement and evaluates it. If it is still true, the same operation is repeated. If it is no longer true, program execution moves to the statement immediately following the WEND statement. WHILE . . . WEND loops can be nested as deeply as the capacity of the memory will allow. Each WEND will correspond to the nearest previous WHILE. A WHILE without a WEND will cause a WE error and a WEND without a WHILE will result in a WH error.

# SYSGEN

**Function** To create a new system disk.

**Format** **SYSGEN**

**Example** **SYSGEN**

**Remarks** Before executing the SYSGEN statement, first place the source system disk in drive A and the disk which you wish to make into the new system disk in drive B. The HX-20 will confirm by asking "Are you sure?" At this point, if you wish to execute the statement, input "Y" from the keyboard, and if you wish to cancel execution, input "N". As the SYSGEN statement copies only the system area and files whose file type is "SYS", it has no effect on other files in the disk. This makes it possible to change a non-system disk into a system disk. If your version of disk BASIC has changed, you can upgrade the earlier system disk by copying the new system disk onto it.

To make disks that were used with another system into system disks, execute a FRMAT statement before executing the SYSGEN statement. If the disk set in drive B is different from the disk previously set in drive B, a "Disk write protected" error will occur if you do not execute a RESET statement after setting the new disk. If the disk set in drive A is not a system disk, an NE error will occur. Also, if the disk set in drive B does not have enough disk or directory space to copy the program file, a "Directory full" or "Disk full" error will occur.

# CVI, CVS, CVD

**Function** To convert string data back to numeric data.

**Format**
**CVI <2-byte string>**
**CVS <4-byte string>**
**CVD <8-byte string>**

**Example**
```
A% = CVI (A$)
B! = CVS (B$)
C# = CVD (C$)
```

**Remarks** Those functions reconvert numeric data converted to string by MKI$, MKS$ and MKD$.
CVI converts a 2-byte string into an integer, CVS converts a 4-byte string into a single precision number and CVD converts an 8-byte string into a double precision number.
If the length of the string is less than that required for each function. an FC error will occur. If the length of the string is greater than that required, only the necessary length will be taken from the beginning of the string.

(See MKI$, MKS$, MKD$.)

# DSKF

**Function** To return the free area remaining on the disk.

**Format** **DSKF (<drive name>)**

**Example** **PRINT DSKF ("A:")**

**Remarks** DSKF returns the amount of free area in the disk set in the specified disk drive in units of kilobytes.
<drive name> is a string with a value of "A:" through "D:".
If the RESET statement is not executed after the disk is replaced, DSKF will not return a correct value, so caution is advised.
Also, as the memory area for BASIC DOS is allocated in 2K-byte units, called blocks, the value returned by DSKF will change in multiples of two.

# DSKI$

**Function** To read data directly from the disk.

**Format** **DSKI$ (<drive name>, <track No.>, <sector No.>)**

**Example** **A$=DSKI$ ("A:", 1, 3)**

**Remarks** DSKI$ reads one logical sector from the specified drive and track and returns the result as a 128-byte string.
The specification of <drive name>, <track No.> and <sector No> is the same as DSK0$ and, as with DSK0$, there is no need for opening or closing.

(See DSK0$.)

# EOF

**Function** To return the end-of-file code.

**Format** **EOF (<file number>)**

**Example** **IF EOF (1) THEN CLOSE #1 ELSE GOTO 100**

**Remarks** The file specified by <file number> must have been opened for the input mode. EOF checks if the file specified by <*file* number> has reached its end.
EOF returns -1 (true) if the end of the file has been reached and returns 0 (false) if not.
If the specified file is RS-232C port ("COM0:"), EOF returns -1 when the buffer is empty and returns 0 when the buffer is not empty. The EOF function always returns 0 (false) for the file assigned to the keyboard.

# INPUT$

**Function** To return a string of characters read from a specified file.

**Format** **INPUT$ (<number of characters> [,[#]<file number>])**

**Example** **A$=INPUT$ (5,** #3)

**Remarks** INPUT$ reads a string of characters in the number specified by <number of characters> from the file specified by <file number>. If <file number> is omitted, characters can be input from the keyboard; but the characters input from the keyboard are not displayed on the screen, unlike the execution of an INPUT statement.
INPUT$ is in a wait state until a string of characters specified by <number of characters> is all input. However, if any input data exists in the input buffer, INPUT$ reads characters from the buffer.
With an INPUT$, all characters except BREAK key are read as is. Therefore, INPUT$ allows the input of characters, such as Carriage Return (character code 13), etc., which cannot be entered by INPUT and LINE INPUT statements.

# LOC

**Function** To return the current record No.

**Format** **LOC (<file No.>)**

**Example** **PRINT LOC (1)**

**Remarks** If the file specified in <file No.> was opened in the R mode, the record No. of the record used in the last GET or PUT statement is returned. If the file was opened in the I mode, the number of logical sectors (128-byte units) read so far is returned. If the file was opened in the 0 mode, the number of sectors written so far is returned.

(See OPEN.)

# LOF

**Function** Returns the largest record No. in the file.

**Format** **LOF (<file No.>)**

**Example** **IF R>LOF (1) THEN 100**

**Remarks** LOF returns the largest record No. written into the specified file.

# MKI$, MKS$, MKD$

**Function** To convert numeric data to string type.

**Format**
**MKI$ (<integer expression>)**
**MKS$ (<single precision expression>)**
**MKD$ (<double precision expression>)**

**Example**
**A$=MKI$ (A%)**
**B$=MKS$ (B!)**
**C$=MKD$ (C#)**

**Remarks** All data stored in the random file buffer must be string type. Therefore, to store numeric data in the random file buffer, this data must first be converted to string type.
MKI$ converts an integer into a 2-byte string, MKS$ converts a single precision number into a 4-byte string and MKD$ converts a double precision number into an 8-byte string. These functions convert numeric data directly into character codes. For example, execution of

```
10 A%=33
20 X=VARPTR (A%)
30 A$=CHR$(PEEK(X))+CHR$
(PEEK(X+1)
```

will yield the same result as

```
10 A%=33
20 A$=MKI$(A%)
```

These functions can be used independent of file operations.

(See CVI, CVS, CVD.)

## 4.6 COPY Utility

To protect your important programs and files against loss due to misoperation or accident, it is always a good idea to make backup copies of such disks. This is a utility program, supplied on the system disk, which allows you to copy files, either by copying the entire disk or by copying only a specific file.

### Functions of the COPY utility

• To copy the entire disk
To copy the entire contents of the disk set in drive A to that set in drive B (or from drive C to drive D).
• To copy files
To copy a single file from the disk selected by the user to specified drive.

### Using the utility

(a) Loading and selection
© Load the utility using one of the following:

```
LOAD"COPY. UTL"
RUN "COPY. UTL"
```

or,

```
RUN "COPY. UTL"
```

② The following will be displayed on the screen. Select whether you wish to copy a disk or a file.

```
Copy utility V-1.0
Copyright 1982
          By EPSON
 Volume copy -- 1
 File copy   -- 2
 End         --^C
Select(1/2/^C)?
```

Screen 1

To copy a disk, input 1, to copy a file, input 2, and to terminate the execution of the utility, input CTRL + C.

(b) Copying disks

© Select "1" from the screen displayed in Screen 1 above.

@ The following will be displayed asking you to make your selection of disk drives.

```
Volume copy
 From A: To B: -- 1
 From C: To D: -- 2
Select(1/2/^C)?
```

Screen 2

To copy from drive A to drive B, input 1, from C to D, input 2. If **CTRL** + **C** is input, the program will return to the screen displayed in Screen 1 and you will again be asked to select between copying a disk and copying a file.

③ If you input either 1 or 2, the following will be displayed. Check the drive, insert the disk and press the **RETURN** key. Copying will now begin.

```
Volume copy
 From A: To B:
Change the diskette!
Press RETURN or ^C
```

Screen 3

At this point, if you wish to cancel the copy processing, press **CTRL** + **C** and the program will return to the display shown in Screen 1.

④ When copying has finished, the following message will be displayed and control will return to the state shown in Screen 1.

```
Finished!
```

Screen 4

(c) Copying files

① Select file copy from the screen display described in Screen 1.

② The following will be displayed asking you to input the file descriptor of the source file and the file descriptor of the destination file.

```
File copy
 From: A:SRCE.TST
 To: B:DEST.TST
```

Screen 5

(The underlined portions are to be supplied by the user.)

The format of the file descriptor is the same as in ROM BASIC. If the device name is omitted, drive A will be assumed. Copying can be performed from any disk drive to any disk drive the user chooses. If only the drive name is specified after "To:", the source file will be copied in the new disk with the same filename. If a filename is specified, the source file will be copied to the disk with the specified name.

If you press only the RETURN key in place of the file descriptor, the program will return to the display shown in Screen 5.

③ When you have finished inputting the filename, the computer will ask for confirmation. At this point, if you wish to execute the copying operation, press the RETURN key. If you wish to cancel, press **CTRL** + C to return to Screen 5 and input the correct filename.

If you press the RETURN key, file copying will begin.
When copying is finished, the following message will be displayed and the program will return to Screen 5.

Finished!

Screen 6

## 4.7 Error Processing

(1) Error during OPEN

Remove the cause of the **error and then execute another OPEN** statement.

(2) Error during output

If an error such as DU, "Disk write protected", "Disk read" or "Disk write" occurs when using a PUT or PRINT# statement, close all open files immediately. If you continue to output, the contents of the files are not protected against possible loss.

(3) Error during input

If an error such as DU or "Disk read" occurs when executing a GET or INPUT# statement, close all open files immediately. Again, the contents of the files cannot be guaranteed if you continue to input.

(4) "Disk write protected"

The only processing for this error is to execute a RESET command. However, a RESET command will destroy all of the contents of any file so unfortunate as to be open at the time of execution. Therefore, before executing a RESET command, continue to execute parameter-less CLOSE statements (i.e., CLOSE statements that do not specify a particular file but close all the files on the disk) until the error message is no longer displayed.

(5) DU

This error occurs when an attempt is made to access a drive which does not contain a disk. The TF-20 containing that drive will continue to output this error until a disk is inserted in the drive.

*NOTE:*

*There may be instances when a READ error occurs during execution of a PUT statement. This is because the previous GET statement has not been correctly executed. Similarly, a WRITE error may occur during execution of a GET statement because of incorrect execution of the previous PUT statement.*

# 5. HARDWARE

The TF-20 is an intelligent flexible disk unit incorporating a CPU, a 64K-byte memory and I/0's. Therefore, it lightens the workload for file management of the CPU in the host computer and reduces the amount of memory space in the host computer to be occupied by the operating system (OS). The TF-20 features the following:

(1) Incorporates a high-speed serial interface for connection with the HX-20.
(2) Performs all file management by transferring necessary information to and from the HX-20.

## 5.1 Hardware Configuration

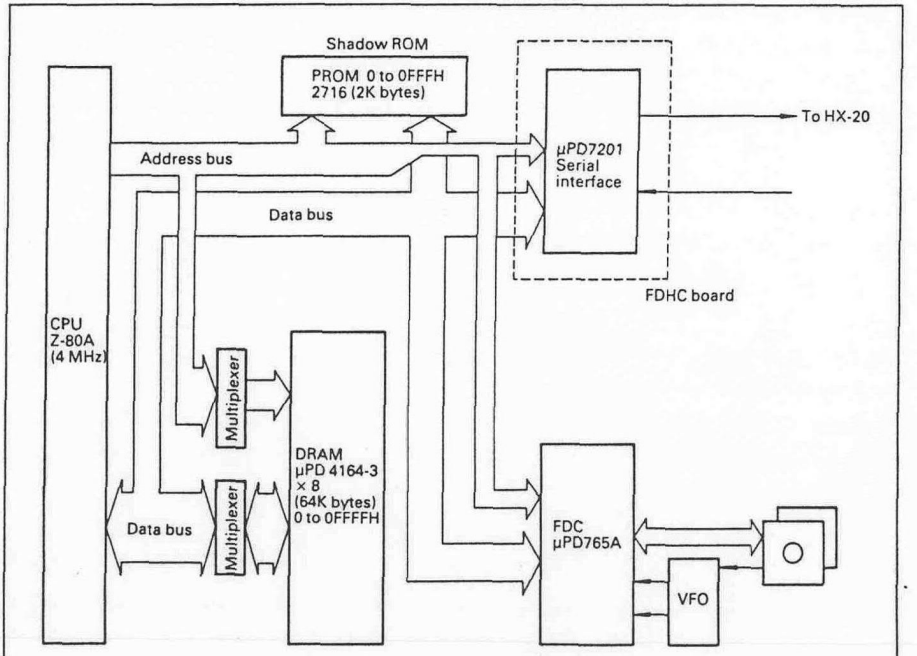



Fig. 5.1 Block Diagram

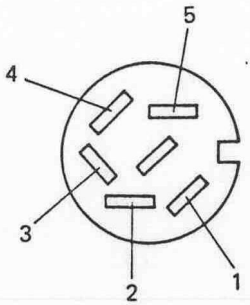
## 5.2 Interface Signals

### (1) Interface connector to the HX-20

Interfacing with the HX-20 is carried out **by daisy chaining using cable set** #707.

Connector name: DIN 6 Pin (female)

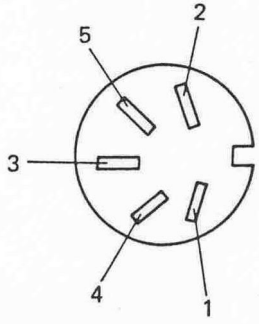
| Pin No. | Signal name | Direction (as viewed from Flexible Disk Unit) | Description |
|---|---|---|---|
| 1 | RXS | OUTPUT | Output signal from the TF-20. This signal is connected to other terminals or the serial input of the HX-20. |
| 2 | PINS | OUTPUT | OR signal between the PIN0 terminal and the DTRA terminal of internal serial controller uPD7201. Usually, this signal is not used. |
| 3 | TXS | INPUT | Serial input signal from the HX-20 to the TF-20. |
| 4 | POUTS | INPUT | Input signal. This signal is connected to the CTSA terminal of internal serial controller uPD7201 through the line receiver. "HIGH" level of this signal enables serial output. |
| 5 | GNDS | — | Ground |



### (2) Interface Connector to Other Terminals

Connector name: DIN 5 Pin (female)

| Pin No. | Signal name | Direction (as viewed from Flexible Disk Unit) | Description |
|---|---|---|---|
| 1 | GNDC | — | Ground |
| 2 | TXC | OUTPUT | Serial signal output from the TF-20. |
| 3 | RXC | INPUT | Input terminal. When the level of this signal is "LOW", signals from the TF-20 will be output from RXS. |
| 4 | POUTC | OUTPUT | The same as POUTS signal. (Not used.) |
| 5 | PINC | INPUT | OR signal between this signal and the DTRA of serial controller µPD7201 is output to PINS. (Not used.) |




## 5.3 Interface Level

RS-232C serial interface: Logic 1 (-3 to -27V)
Logic 0 (+3 to +27V)

*NOTE:*
*The interface on the rear panel of the TF-20 is not* **used.**

# APPENDIX A ERROR CODES

For Disk BASIC, the following error codes have been added.

| Error code | Error message |
|---|---|
| 64 | **Directory full**<br>The disk is full and new files cannot be created.<br>Each file uses a directory for every 32K bytes so this error can also result from file expansion. |
| 65 | **Too many open files**<br>More disk files than were specified as the number of FCBs in the FILNUM statement are being opened. |
| 66 | **Disk full**<br>The user is attempting to expand the files even though all the disk space is being used. |
| 67 | **File already exists**<br>The filename specified as a new file in the NAME command already exists on the disk. |
| 68 | **Field overflow**<br>The user has attempted to assign variable lengths which total to more than the 128-byte maximum permitted in the FIELD statement. |
| 69 | **Bad record number**<br>0 has been used as a record No. in a GET or PUT statement. |
| 70 | **Disk write protected**<br>The user has attempted to write on a disk with a write protect label.<br>The user has attempted to write on a disk that has been replaced without first executing a RESET command. |
| 71 | **READ**<br>May occur during execution of PUT. Refer to Error Processing. |
| 72 | **WRITE**<br>May occur during exeption of GET. Refer to Error Processing. |

# APPENDIX B DISKDRIVE ENVIRONMENTAL CONDITIONS

**Temperature**

| | |
|---|---|
| Operating state | : 5 to 28°C |
| Non-operating state | : -30 to 65°C |

**Humidity**

| | |
|---|---|
| Operating state | : 20 to 80% (non-condensing) |
| Non-operating state | : 5 to 85% (non-condensing) |

# APPENDIX C POWER REQUIREMENTS AND OUTLINE DIMENSIONS

**Power Requirements**

Power Frequency : 49.5 to 60.5 Hz

Power Consumption : 40W

**Outline Dimensions and Weight**

Outline Dimensions : 120(W) mmx350(D)mmx165(H)mm

Weight : 6kg

# APPENDIX D DISK SPECIFICATIONS AND ENVIRONMENTAL CONDITIONS

| | |
|---|---|
| Format | : Double-sided, double density |
| No. of tracks | : 80 tracks (40 tracks x2) |
| Track density | : 48 TPI |
| No. of sectors | : 16 sectors/track |

**Temperature**

| | |
|---|---|
| Operation | : 10to50°C |
| Storage | : 4 to 53°C |
| Transport | : -50to53°C |

**Humidity**

| | |
|---|---|
| Operation | : 20 to 80% RH (Max. wet bulb temperature: 29°C) |
| Storage | : 8 to 80% RH |
| Transport | : 8 to 90% RH |

## EPSON OVERSEAS MARKETING LOCATIONS


### EPSON AMERICA, INC.
3415 Kashiwa Street
Torrance, CA 90505 U.S.A.
Phone: (213)539-9140
Telex: 182412

### EPSON UK LTD
Dorland House
388 High Road,
Wembley, Middlesex, HA9 6UH. U.K.
Phone: (01) 900-0466/7/8/9
Telex: 8814169

### EPSON DEUTSCHLAND GmbH
Am Seestern 24
4000 Düsseldorf 11
F.R Germany
Phone: (0211)596-1001
Telex: 8584786

### EPSON ELECTRONICS (SINGAPORE) PTE. LTD.
#09-13/#09-14 World Trade Centre,
No. 1 Maritime Square, Singapore 0409
Phone: 2786071/2

### EPSON ELECTRONICS TRADING LTD.
Room 411, Tsimshatsui Centre.
East Wing 6, Chimg Yee Road
Tsimshatsui Kowloon, Hong Kong
Phone: 3694343/4
3-7213427
3-7214331/3
Telex: 64714

### EPSON ELECTRONICS TRADING LTD. TAIWAN BRANCH
1.81F K.Y. Wealthy Bldg. 206, Nanking
E. Road, Sec. 2. Taipei, Taiwan, R.O.C.
Phone: 536-4339
536-3567