PORTABLE COMPUTER

HX-20

BASIC REFERENCE MANUAL



EPSON

H8294012

NOTICE:

* All rights reserved. Reproduction of any part of this manual in any form whatsoever without EPSON's express written permission is forbidden.
* The contents of this manual are subject to change without notice.
* All efforts have been made to ensure the accuracy of the contents of this manual. However, should any errors be detected, EPSON would greatly appreciate being informed of them.
* The above notwithstanding, EPSON can assume no responsibility for any errors in this manual or their consequences.

© Copyright 1982 by EPSON CORPORATION.
Nagano, Japan

FOREWORD

The EPSON HX-20 Portable Computer is the ultimate in personal computers – a complete desktop personal computer system miniaturized to fit in a briefcase. To take full advantage of this portability, the hardware and software features listed below have been incorporated in its design.

This manual provides a detailed explanation of one of those features, the unique EPSON developed BASIC, and it is hoped that the user will profitably use this volume as a reference for programming as well as for further application software development. To those ends, we have edited and prepared this manual with ease of reference and use in mind.

Features

1. The RAMs are backed up with batteries so that programmes and data stored in the RAM are protected against loss even when the power switch is turned OFF. Programme execution is possible upon power application.
2. The memory space is divided into five programme areas, each capable of storing a separate BASIC programme. Each programme can be selected from the menu for immediate execution.
3. The 5 programmes in the menu are managed independently, so that the creation of a new programme or the editing of the existing programme does not affect other programmes in the memory.
4. The HX-20 features an 80-character (20 char. by 4 lines) liquid crystal display (LCD). A virtual screen for larger than the LCD screen can be specified by a WIDTH command and the capacity to scroll freely in any of the four directions enables the user to perform screen editing of the large internal screen beyond the physical limitations of the LCD.
5. All operations of the optional microcassette drive can be controlled under BASIC.
6. In addition to the programme areas, the HX-20 is provided with a RAM file area to facilitate data storage, as well as data transfer between programmes.
7. By connecting an optional TF-20X terminal floppy unit to the HX-20, DISK BASIC can be run.

TABLE OF CONTENTS

Chapter 1 Introduction to HX-20

1.1 Initialisation and BASIC 1-1
1.2 Operation of keyboard 1-6
1.3 Text and graphic screens 1-10
1.4 Text screen (virtual screen) 1-12
1.5 Screen edit 1-14
1.6 File operations 1-16
1.7 Peripheral equipment 1-18

Chapter 2 Outline of EPSON BASIC

2.1 Operation modes 2-1
2.2 Statements 2-2
2.3 Lines 2-2
2.4 Character set 2-2
2.5 Constants 2-3
2.6 Variables 2-4
2.7 Type conversion 2-5
2.8 Expressions and operations 2-6
2.9 Error messages 2-11
2.10 DIP switch setting 2-12

How to use Chapters 3 and 4 2-13

Chapter 3 Commands and Statements

Chapter 4 Functions

Chapter 5 Additional Information

5.1 RAM files 5-1
5.2 Sequential files 5-5
5.3 Machine language programmes 5-9
5.4 How to use the RS-232C port 5-14

APPENDIXES

A. Error Messages
B. Device Names
C. Correspondence Table between Device Names and EPSON BASIC Commands
D. Formatting Characters
E. Keyboard Layouts and Key assignments
F. Character Code Tables
G. Control Codes
H. Memory Map
I. Table of Reserved Words
J. List of Commands and Statements

Commands and Statements

AUTO 3-1
CLEAR 3-2
CLOSE 3-3
CLS 3-3
COLOR 3-4
CONT 3-5
COPY 3-5
DATA 3-7
DEFFIL 3-8
DEF FN 3-9
DEFINT/SNG/DBL/STR 3-10
DEF USR 3-12
DELETE 3-13
DIM 3-14
END 3-14
ERASE 3-15
ERROR 3-16
EXEC 3-17
FILES 3-18
FOR...TO...STEP – NEXT 3-19
GCLS 3-21
GET% 3-21
GOSUB – RETURN 3-22
GO TO/GOTO 3-23
IF...THEN...ELSE/
IF...GOTO...ELSE 3-24
INPUT 3-25
INPUT# 3-26
KEY 3-26
KEY LIST/KEY LLIST 3-27
LET 3-27
LINE 3-28
LINE INPUT 3-29
LINE INPUT# 3-30
LIST/LLIST 3-30
LIST<file descriptor> 3-31
LIST"COM0:" 3-31
LOAD 3-32
LOAD"COM0:" 3-32
LOADM 3-33
LOAD? 3-34
LOCATE 3-35
LOCATES 3-36
LOGIN 3-36
MEMSET 3-37
MERGE 3-38
MERGE"COM0:" 3-38
MID$ 3-39
MON 3-40
MOTOR 3-41
NEW 3-42
ON ERROR GOTO 3-42
ON...GOSUB/ON...GOTO 3-44
OPEN 3-45
OPEN"COM0:" 3-46
OPTION BASE 3-48
PCOPY 3-49
POKE 3-50
PRESET 3-52
PRINT/LPRINT 3-53
PRINT USING/LPRINT USING 3-54
PRINT# 3-58
PRINT# USING 3-58
PSET 3-59
PUT% 3-60
RANDOMIZE 3-61
READ 3-61
REM 3-62
RENUM 3-62
RESTORE 3-63
RESUME 3-64
RUN 3-65
RUN"COM0:" 3-65
SAVE 3-66
SAVE"COM0:" 3-67
SAVEM 3-67
SCREEN 3-68
SCROLL 3-69
SOUND 3-70
STAT 3-71
STOP 3-72
SWAP 3-72
TITLE 3-73
TRON/TROFF 3-74
WIDTH 3-75
WIDTH<device name> 3-76
WIND 3-77

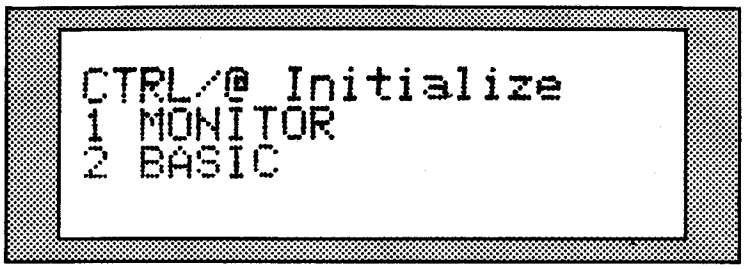
Functions

ABS 4-1
ASC 4-1
ATN 4-2
CDBL 4-2
CHR$ 4-3
CINT 4-3
COS 4-4
CSNG 4-4
CSRLIN 4-5
DATE$ 4-5
DAY 4-6
EOF 4-6
ERL/ERR 4-7
EXP 4-7
FIX 4-8
FRE 4-8
HEX$ 4-9
INKEY$ 4-10
INPUT$ 4-11
INSTR 4-12
INT 4-13
LEFT$ 4-14
LEN 4-14
LOF 4-15
LOG 4-15
MID$ 4-16
OCT$ 4-17
PEEK 4-17
POINT 4-18
POS 4-19
RIGHT$ 4-19
RND 4-20
SGN 4-21
SIN 4-21
SPACE$ 4-22
SPC 4-22
SQR 4-23
STR$ 4-23
STRING$ 4-24
TAB 4-25
TAN 4-25
TAPCNT 4-26
TIME$ 4-27
USR 4-27
VAL 4-28
VARPTR 4-28

CHAPTER 1
Introduction to HX-20

1.1 Initialisation and BASIC

Unlike other personal computers, the memory space of your HX-20 Portable Computer is divided into 5 programme areas where 5 BASIC programmes can be stored independently. These programmes will be retained in the memory without any change even if you turn OFF the power switch of the HX-20. The HX-20 incorporates a menu function to enable any of the stored programmes to be executed with a single-touch key operation. Turn ON the power switch, and the menu will appear on the LCD screen as follows.



The menu displays numbers and the functions (i.e., programme names) which can be executed when you press the corresponding numeric keys. When power is applied to the HX-20 for the first time, only two functions will appear on the LCD screen as shown above. "CTRL/@" at the uppermost line of the display indicates that you must press the @ key while holding down the CTRL key.

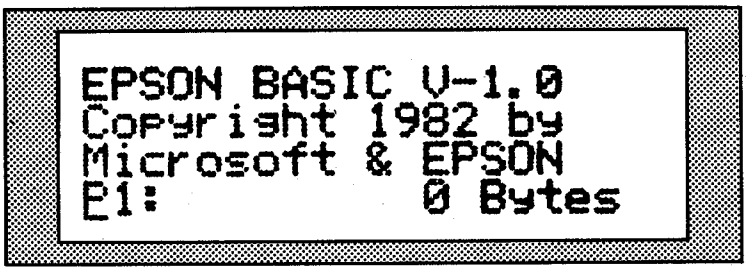
NOTE:

If you select a character set other than that for USA, England, Italy or Spain, the following symbols will appear in lieu of "@".

USA	France	Germany	Sweden
@	à	§	É

The character code and function of each of these keys is the same as "@" and the corresponding character for each character set should be pressed while holding down CTRL *key. (See Section 2.4, Character Sets, for details.)*

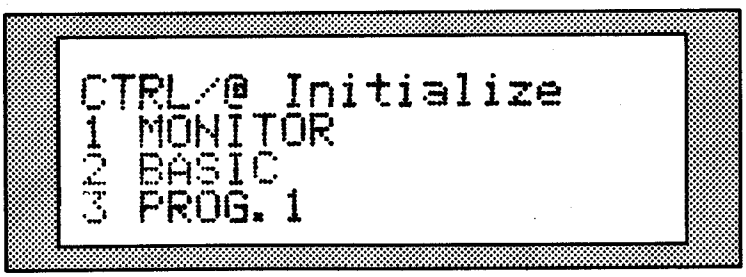
If you do this followed by the date and time setting described later, your HX-20 will be initialised. This state is called "cold start". (You must initialise your HX-20 by cold start when you turn ON the power switch for the first time after purchase.) To execute EPSON BASIC, press numeric key "2".



When BASIC is executed, the programme area No. 1 is always selected. The message "0 Bytes" following "P1:" indicates the length of the programme stored in the current programme area. In this case, as no programme has been stored in program area No. 1, the value displayed is "0" bytes.

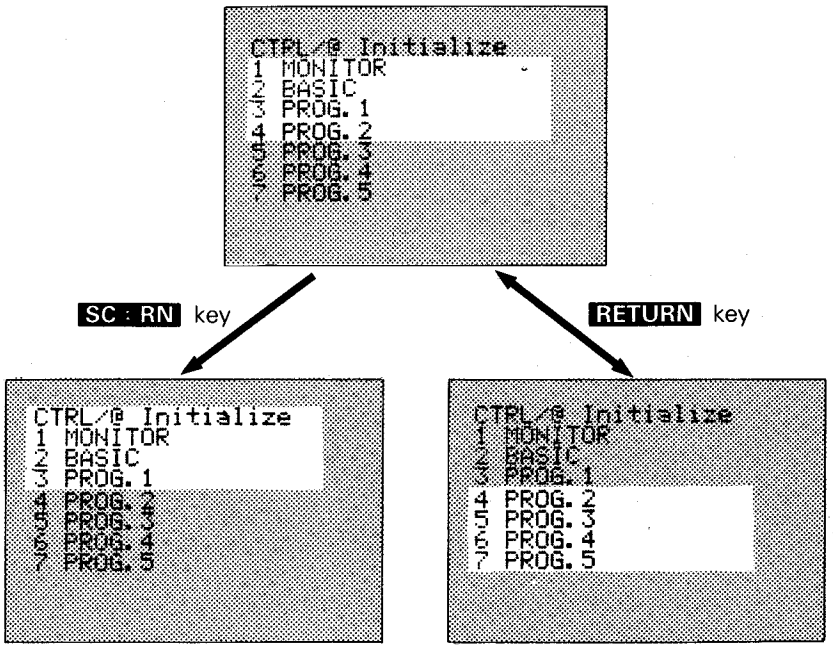
1.1.1 Menu

If you name a programme that you have written in BASIC with a TITLE command, the programme name will appear in the menu on the display. This means that the programme name has been registered in the menu.



5 programmes registered in the menu are numbered 3 through 7, respectively. Any of these programmes will be executed directly (without using LOGIN and RUN commands) whenever you press the numeric key corresponding to the programme number.

When you call the menu after all the programme names have been registered, the screen will scroll up to show you the programme names gradually as all the programme names cannot be displayed at one time in the LCD display window. After the display of all the programme names, it will then return to the first name "1 MONITOR", indicating that the HX-20 system has returned to command level. If you press **RETURN** key at this moment, the current display is switched with the display outside the physical screen to show you the remaining programme names. Pressing **SC : RN** key will return the display to the initial screen. If you press **SHIFT** and **SC : RN** keys, the screen will move backwards.

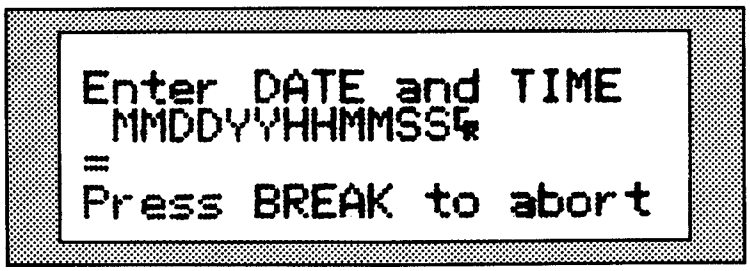


NOTE:
The numbers assigned to the programmes are used only for menu display and the K command of the MONITOR, and have no direct connection with the programme area numbers used by a LOGIN command. Also, these numbers may change upon registration of other programmes in the menu.

The K command of the MONITOR makes the HX-20 ready for direct programme execution without requiring any other operation immediately after the power switch is turned ON.

1.1.2 Cold start

When CTRL key and "@" key are pressed while the menu is being displayed, the HX-20 will ask you to input the date and time.



At this point, if you type the current date in the format "MMDDYY" (month, day and year) and the current time in the format "HHMMSS" (hours, minutes and seconds) in 12 digits (24-hour system), and then press RETURN key, the HX-20 will be initialised. This state is called "cold start". You can correct the time setting by pressing INS DEL key. When the HX-20 is initialised by cold start, all the memory contents of the HX-20 are cleared and the respective constants in BASIC are reset as follows. (For details, refer to the applicable chapters.)

```
CLEAR 200,256
MEMSET &H0A3F
KEY  1, "AUTO"
KEY  2, "LIST" + CHR$(13)
KEY  3, "LLIST" + CHR$(13)
KEY  4, "STAT"
KEY  5, "RUN" + CHR$(13)
KEY  6, "?DATE$:?TIME$" + CHR$(13)
KEY  7, "LOAD"
KEY  8, "SAVE"
KEY  9, "TITLE"
KEY 10, "LOGIN"
```

As previously described, when the HX-20 is cold started, all the data currently stored in the HX-20 memory are cleared. So please be careful not to cold start your HX-20 unnecessarily. If you accidentally press CTRL and @ keys, do not set the date and time but press BREAK or MENU key. This will cause the HX-20 to return to the menu display again.

1.1.3 Warm start

All the execution of BASIC by other than the cold start is called "warm start". When the HX-20 is initialised by warm start, the respective constants in BASIC are reset as follows. (These are called "default values after warm start".)

```
CLEAR 200
SCREEN 0,0
SCROLL 9,0,10,4  } LCD
WIDTH 40,8,3     }
SCROLL 9,0,16,16 } External display
WIDTH 40,37,5    }
```

If you press **MENU** key even during the execution of a BASIC programme, BASIC will enter the warm start state. Therefore, please note that while you are operating the HX-20, for example, to change the screen size, the default values will be assumed after the menu has been displayed by pressing **MENU** key. (The size of the external display, however, will not be affected by warm start.

1.1.4 Reset

There is a RESET switch, recessed at the rear of the right-hand side of your HX-20. This switch need not be operated during normal BASIC operation. Never use the RESET switch except in the circumstances as described below. The only times when you must operate the RESET switch are those cases where the HX-20 does not respond to **BREAK** key and the POWER switch (as a result of a programme overrun). In the HX-20, a programme overrun will never occur unless the HX-20 is operated incorrectly as follows.

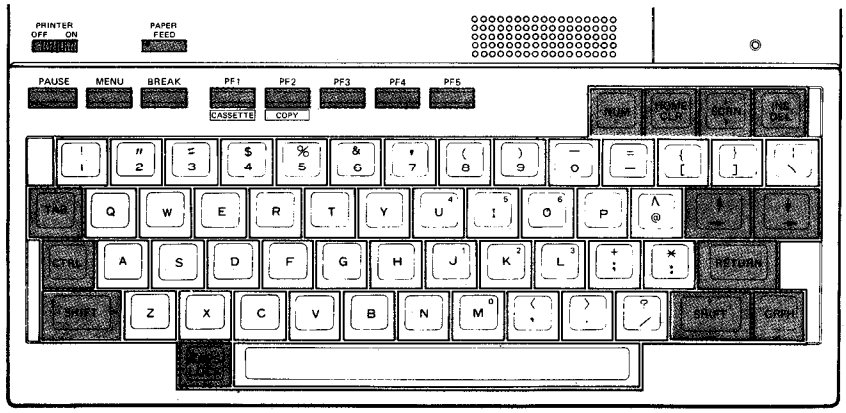
- Data rewrite in the system area (memory addresses &H004D to &H0A3F)
- Attempt to read the data in the I/O area (memory addresses &H0000 to &H004D).
- Execution of an incompletely written machine language programme.

If a programme overrun should occur as a result of one of these operations, turn the POWER switch OFF immediately and then press the RESET switch.
When you attempt to execute BASIC after depressing the RESET switch, there may be a case where part of the memory has been rewritten due to overrun. In such a case, the menu display is disrupted, and your attempts to execute BASIC are not successful. Should this happen, you must initialise the HX-20 by cold start to clear all the memory contents of the HX-20. Before executing an incompletely written machine language programme, always save the data and programmes in the memory to a cassette tape or other external memory so that even if the memory should be erased, no problem will occur.

NOTE:
While you are operating the HX-20, there may be a case where the message "CHARGE BATTERY!" will suddenly appear on the LCD screen. If this happens, all the HX-20 operations will be suspended and the HX-20 will not accept any inputs. This is a warning message to tell you that the battery voltage is low. If the battery voltage of the HX-20 falls below a certain level, the HX-20 will stop its operation to protect the programmes and data in the memory. If this message appears, turn the power switch OFF and recharge the batteries as soon as possible.

1.2 Operation of keyboard

The keyboard arrangement of the HX-20 is as shown below.



NOTE:
The keyboard arrangement shown above applies to that for U.S.A. Refer to APPENDIX E for England Keyboard.

1.2.1 Keyboard modes

The HX-20 has three modes for the input of characters. The characters that can be input in each of these modes is different.

(1) Uppercase Mode

In this mode, the normal keyboard input is uppercase characters. For example, by pressing the key marked "A" on the keyboard, the character "A" will be input. If this key is pressed while holding down SHIFT key, lowercase "a" will be input. For the numeric and symbolic keys, the number or symbol appearing on the lower half of the key will normally be input and the number or symbol on the upper half of the key will be input when the key is pressed while holding down SHIFT key.

For example, when [: *] key is pressed in the uppercase mode, ":" is normally input and "*" is input if this key is pressed while holding down SHIFT key. The uppercase mode is the default mode and the HX-20 is set in this mode every time BASIC is executed.

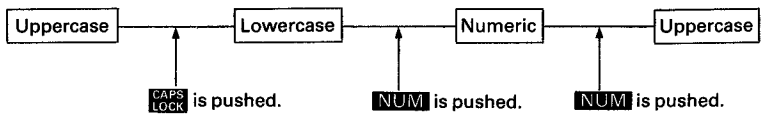
(2) Lowercase Mode

The HX-20 enters this mode when the key marked CAPS LOCK is pressed. In this mode, lowercase letters are normally input and uppercase letters are input by pressing a letter key while holding down SHIFT key. The function of the numeric and the symbolic keys is the same as in the uppercase mode.

(3) Numeric Mode

The HX-20 enters the numeric mode when the NUM key is pressed. The keys which can be operated in this mode are the numeric keys at the top of the keyboard and the letter keys which have the numbers 0 to 6 written in their upper right-hand corner as well as the symbolic keys (+), (−), (*), (/), (.), (,) and (?). All other keys will be ineffective even if you press them. For the characters assigned for each mode, refer to APPENDIX E, "Key Assignments for Each Keyboard Mode".

You can change the mode by pressing one of the keyboard mode keys. However, if you press the key for the mode which the HX-20 is currently in, the keyboard mode will return to the uppercase mode. An example of changing key modes is shown below.



Keys marked [ ] in the Figure on page 1-6 are special function keys and can be input in any mode.

1.2.2 Input of graphic characters

Graphic characters can be input in the uppercase mode by pressing the respective keys while holding down the GRPH key. For details, refer to APPENDIX E, "Key Assignment for Each Keyboard Mode".

1.2.3 Special functions of CONTROL key

CTRL key, when used in conjunction with other keys, inputs control codes that perform the special functions such as the movement of the cursor on the screen, etc. (For details of these control codes, see APPENDIX G.)
In addition, CTRL key has two special functions.

CTRL + PF2 Copies the data displayed on the LCD on the built-in microprinter. This function is the same as a COPY command execution.

CTRL + PF1 Sets the optional microcassette drive in manual operation mode. This mode will not be entered if the microcassette is not connected. When the manual operation mode is entered, the display is extinguished and the tape counter value is displayed in the upper right-hand corner of the physical screen. In the manual operation mode, the microcassette operations can be controlled by the programmable function keys PF1 to PF6 as follows.

PF1 Fast forward.

PF2 Slow forward.

PF3 Stops the tape rewind, fast forward or slow forward.

PF4 Rewinds the tape.

PF5 Causes exit from manual operation mode.

PF6 Resets the tape counter value.
(PF6 is input by pressing PF1 while holding down SHIFT key.)

1.2.4 Functions of special keys

PAPER FEED Used to feed paper into the the built-in microprinter.

PAUSE Used to temporarily stop the programme execution and listing. The interrupted operation is resumed upon pressing any other key on the keyboard. In this case, if one of the numeric keys 0 through 9 is pressed to resume execution, the scrolling speed can be specified.

MENU Used to return the HX-20 to the state prior to BASIC execution. The menu is displayed and the HX-20 waits for input of your selected function.

BREAK	Used to stop programme execution or listing. Programme execution can be resumed by input of a CONT command.
PF1–PF5 PF6–PF10	You can define the special functions by software in these keys. (For details, see KEY command in Chapter 3.)
RETURN	Used to signal BASIC that input of data in the required units has been completed. When this key is pressed, the cursor moves to the beginning of the next line.
NUM GRPH	These keys are used to select the keyboard mode.
TAB ← → SC ↑ RN	These keys are used for screen editing. For details of each key, refer to Section 1.5.

For the built-in microprinter, there is a PRINTER ON/OFF switch and a PAPER FEED key. The microprinter can only be operated when the PRINTER ON/OFF switch is in the ON position. For example, even if you execute an LLIST statement or press the PAPER FEED key, if the PRINTER ON/OFF switch is OFF, the microprinter will not function. The PAPER FEED key will feed the roll paper into the feed only while it is being pressed.

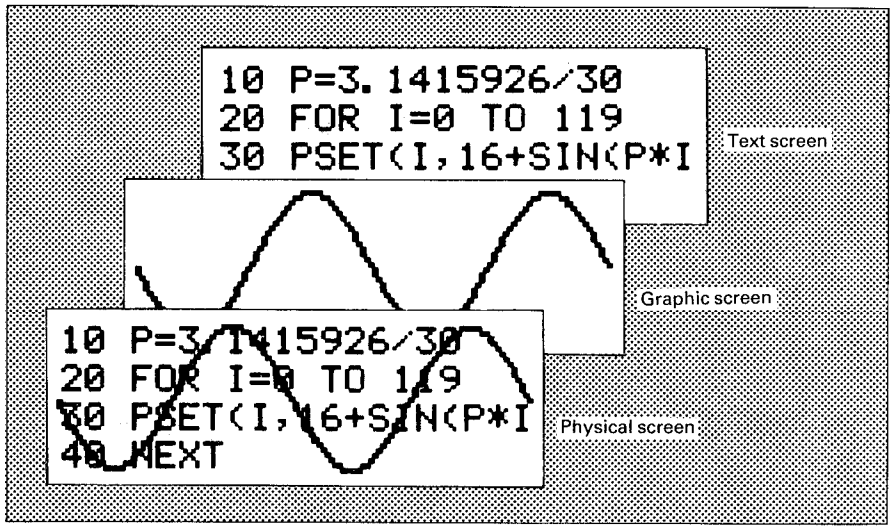
1.2.5 Auto-repeat function

If the alphanumeric keys are pressed continuously, they have a function to automatically input continuously. In addition, the following special keys have the same function.

HOME CLR SC ↑ RN INS DEL ← → TAB RETURN

1.3 Text and graphic screens

The HX-20 has two distinct screens that you may use for the entirely different purposes. The first screen called a "text screen" is to display characters and the second called a "graphic screen" is used to draw graphics using such statements as LINE and PSET. These two screens can be output through two different output devices; one is the built-in LCD and the other is an optional external display.
In EPSON BASIC, with a SCREEN statement, you can specify three different methods of display by combining the two screen modes and the two output devices. When BASIC is executed, both the text screen and graphic screen are specified for display on the LCD.

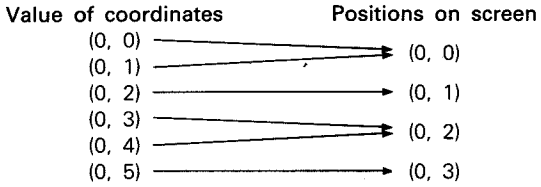


When you want to draw lines and dots on the graphic screen, you must use rectangular coordinates with the upper left-hand corner of the display as the origin. The value of a horizontal coordinate increases as a point on the X-axis moves towards the right, while the value of a vertical coordinate increases as a point on the Y-axis moves downwards. The range of coordinates for the LCD is different from that for the external display. Also, for the external display, the range of coordinates varies depending on the selected display mode. In a LINE or PSET statement, the HX-20 does not check to see whether or not the specified coordinates are actually within the screen. Please pay special attention to the range of coordinates.

	Mode	Horizontal (dots)	Vertical (dots)
LCD		120	32
External display	4-colour	128	64
	High resolution	128	96

NOTE:
The above table indicates the resolution of each display and does not indicate the range of coordinates. For example, the LCD has a resolution of 120×32 dots. But the specified values of coordinates must be in the range of 0 to 119 horizontally and 0 to 31 vertically.

When the external display is set in 4-colour graphic mode, the resolution of the display is 64 dots vertically. But the range of coordinates that you can specify is between 0 and 96, which is the same as that in high resolution mode. In other words, the values of your specified coordinates do not correspond to the dots on the screen.

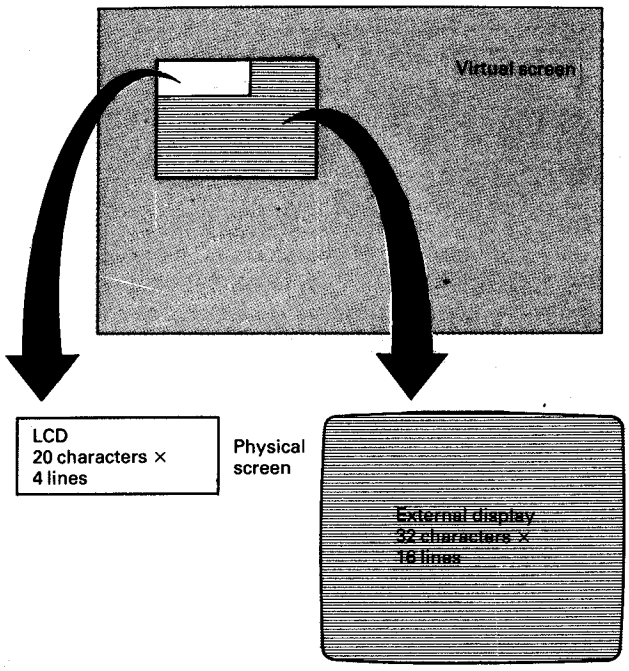


Assume that the value of the vertical coordinate to be specified is N and the vertical position on the screen is V.
The relationship between these two can be expressed by the following formula.

V = 2N\3 ("\" indicates integer division.)

1.4 Text screen (virtual screen)

The LCD screen of the HX-20 is capable of displaying only 80 characters (20 characters×4 lines) at one time. However, by moving the cursor, a new screen may appear one after another. You can operate the HX-20 as if you are using a large screen..
This is because the HX-20 has adopted the concept of a "virtual screen". Let's suppose there is a very large screen inside the HX-20. Commands such as LIST, PRINT are all output to this internal screen. Now, assume the LCD display as a "viewing window" that allows you to see only a part of the internal screen. This "viewing window" can be moved anywhere on the large internal screen. It cannot, however, leave the bounds of the internal screen. The same concept of a "virtual screen" also applies to the external display. The only difference is that the size of the "window" through which you can see is larger than that of the LCD's window. As compared with the virtual screen, this "window" on both the LCD and external display is called the "physical screen".
Thanks to this virtual screen, you can see statements which were previously input or output by moving the physical screen. However, if the virtual screen becomes full of data, the screen will be scrolled up by one line for the next display and thus you cannot see the line overflowed from the virtual screen by moving the physical screen.



The size of the virtual screen can be specified by a WIDTH command. You can specify the size of the virtual screen freely within the range of 255 characters per line or 255 lines, subject to the limitations imposed by the capacity of the memory. (On the external display, the maximum screen size is 40 characters by 37 lines (i.e., 1,480 characters in total). If too large a virtual screen is specified, the available programme area will be restricted. With this in mind, specify the size of the virtual screen as required.
The virtual screen is valid for only the text screen and the size of the graphic screen is fixed.

1.5 Screen edit

EPSON BASIC has a screen edit function to facilitate programme editing. Using this function, a text or programme line being displayed on the screen can be corrected. The corrected text or programme line can be entered into the programme by pressing RETURN key. In other words, you can edit a programme that has already been input and stored in the memory according to the following procedure.

(1) Display the programme to be edited using a LIST command.
(2) Make necessary corrections by moving the cursor.
(3) Press RETURN key.

To enter a text into BASIC, you must first press RETURN key to inform BASIC of the input of the text. BASIC will ignore any and all characters or symbols entered until you press RETURN key. When RETURN key is pressed, BASIC pays attention to only the text being displayed on the line where the cursor is located. For this reason, BASIC makes no distinction between the text called by a LIST statement and the previously entered text or the one displayed by just typing.
The "line where the cursor is located" is not limited to one line currently being displayed on the screen but refers to a text which is continuously typed from the keyboard and does not exceed a maximum of 255 characters in length. BASIC regards a text written over plural lines as a "logical single line". Wherever the cursor is positioned on the "logical single line" when RETURN key is pressed, that line will be input up to its end. When the line on the screen is shorter than the logical single line, you sometimes cannot judge by merely looking at the screen whether the text displayed is part of the logical single line or not. Please pay attention to this point.

1.5.1 Moving the cursor

The following keys are used to freely move the cursor alone during programme editing.

TAB	Moves the cursor to the next TAB position. (Every 8 columns.)
←	Moves the cursor one position to the left.
→	Moves the cursor one position to the right.
SHIFT + ↑	Moves the cursor up one line.
SHIFT + ↓	Moves the cursor down one line.

If the cursor at either end of the physical screen moves out of the physical screen as a result of any cursor movement key operation, the physical screen automatically moves within the virtual screen. In this way, the cursor will be prevented from leaving the physical screen. Also, when warm started, BASIC automatically sets the "scroll margin" to 3. This means that when the cursor reaches the 3rd position from either end of the physical screen, the physical screen will move with the margin left on either end. The scroll margin can be specified with a WIDTH statement.
If the cursor is at either end of a line on the virtual screen, any attempt to move the cursor in that direction will cause it to move to the next (or the preceding) line. The display will, of course, follow any such cursor movement. However, the text advanced to a new line in this way by one of the cursor movement keys will not be considered to be a logical single line. The cursor cannot be moved beyond the upper left- or lower right-hand corner of the virtual screen.

1.5.2 Moving the physical screen

CTRL + A	Moves the physical screen to the left corner of the virtual screen.
CTRL + F	Moves the physical screen to the right corner of the virtual screen.
CTRL + ← or CTRL + S	Moves the physical screen to the left by the specified number of columns.
CTRL + → or CTRL + D	Moves the physical screen to the right by the specified number of columns.
SC ↕ RN or CTRL + P	Moves the physical screen up by the specified number of lines.
SHIFT + SC ↕ RN or CTRL + Q	Moves the physical screen down by the specified number of lines.

These scroll steps are set at the following values when BASIC is warm started: for the LCD, four lines vertically and 10 columns horizontally and for the external display, 16 lines vertically and 16 columns vertically.
The scroll steps setting can be changed using a SCROLL command. The movement of the physical screen differs from that of the cursor. Namely, when the physical screen reaches either end of the virtual screen, it stops there and will move no further.

SHIFT + HOME CLR or CTRL + K	Moves the cursor to its home position which is the upper left-hand corner of the virtual screen.

1.5.3 Insertion and deletion

When correcting a text being displayed on the screen, the character at the cursor position can be changed by typing another character or symbol over it. However, for major corrections, the following key operations are used.

SHIFT + INS DEL or CTRL + R	Upon pressing these keys, the HX-20 enters the insert mode. Any characters typed after this operation will be inserted to the left of the cursor position and the characters to the right of the cursor will move as the cursor moves. This mode continues until you press RETURN key, any of the cursor movement keys, or INS DEL key.
INS DEL or CTRL + H	The characters immediately to the left of the cursor position are deleted and the cursor moves to the left. Mistypes are normally corrected by this operation.
CTRL + E	All the characters to the right of the cursor position on the line where the cursor is currently positioned are deleted. In this case, the cursor does not move.
CTRL + Z	All the characters from the line where the cursor is currently positioned to the bottom line of the virtual screen are deleted. In this case, the cursor does not move.
HOME CLR or CTRL + L	Deletes all the characters on the virtual screen and returns the cursor to its home position.

1.6 File operations

A collection of information is referred to as a "Record" and a collection of records, as a "File". In EPSON BASIC, I/O transfer to and from peripheral devices is performed in units of one record and I/O data is managed in units of one file.
In EPSON BASIC, a special RAM file area is provided.
GET% and PUT% statements can read and write RAM files for efficient data storage and I/O transfer.

1.6.1 File numbers

To enhance the efficiency of I/O, the HX-20 has a special area called "Buffer" in the memory in which records are temporarily stored. The individual numbers assigned to the locations in the buffer area are called "File Numbers". These file numbers are used to access the files.
Each file number must be an integer in the range of 1 to 16.

1.6.2 File descriptors

As the concept of "File" has been adopted for all I/O devices, all I/O operations can be effected by standardized commands. Distinction of one I/O device from another is made by a "file descriptor" which is a string consisting of the following elements.

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

A file descriptor must always be enclosed in double quotation marks.

1.6.3 Device names

<device name> indicates the name of an I/O device and is represented as a rule by a string of either 4 alphabetic characters or of 3 alphabetic characters and 1 numeric character, followed by a colon. The following device names are defined in EPSON BASIC.

Device name	Device	Input	Output	Remarks
KYBD:	Keyboard	○	×	
SCRN:	Screen	×	○	
LPT0:	Built-in microprinter	×	○	
COM0:	RS-232C port	○	○	
CAS0:	Microcassette	○	○	Option
CAS1:	Audio cassette	○	○	
PAC0:	ROM cartridge	○	×	Option
A:	Flexible disk drive A	○	○	Device
B:	Flexible disk drive B	○	○	names used
C:	Flexible disk drive C	○	○	in
D:	Flexible disk drive D	○	○	DISK BASIC

○: Applicable ×: Not applicable

If device name is omitted, EPSON BASIC automatically checks the peripheral equipment connected to the HX-20. If a microcassette drive or a ROM cartridge is connected, BASIC gives precedence to these devices over the audio cassette. If none of these devices is connected, the audio cassette is set. The device set when device name is omitted is called the "default device".

1.6.4 Filenames

<filename> is the name given to each file by the user. No filename can be omitted if I/O data transfer is to be performed between the HX-20 and another auxiliary memory unit such as a cassette or ROM cartridge. In other cases, it can be omitted.

The filename must be used in the following format.

<filename>[.[<filetype>]]

The first <filename> consists of an 8-character string and <filetype> following a period consists of a string of 3 characters. Any characters other than colons, periods, brackets, and character codes 0 and 255 can be used as <filename> and <filetype>.
Usually, <filename> indicates the name of a file and <filetype>, its attribute. This, however, is left to the discretion of the user. <filetype> is normally included in any reference to <filename>.
If you specify a filename exceeding 8 characters or a filetype exceeding 3 characters, an "FD" error occurs.

1.7 Peripheral equipment

The concept of "File" has been adopted to handle the I/O operation between the HX-20 and all peripheral devices and for this reason, use of standardized commands is effective for almost all the devices.
The following table shows the basic commands and statements for handling files and the devices for which the respective commands and statements are valid.

Command \ Device	KYBD:	SCRN:	LPT0:	COM0:	CAS0:	CAS1:	PAC0:
LOAD	×	×	×	○	○	○	○
LOADM	×	×	×	×	○	○	○
LOAD?	×	×	×	×	○	○	×
RUN "<file descriptor>"	×	×	×	○	○	○	○
MERGE	×	×	×	○	○	○	○
FILES	×	×	×	×	○	○	○
INPUT#	○	×	×	○	○	○	○
INPUT$	○	×	×	○	○	○	○
EOF	–	×	×	○	○	○	○
LOF	–	×	×	○	–	–	○
SAVE	×	○	○	○	○	○	×
SAVEM	×	×	×	×	○	○	×
LIST	×	○	○	○	○	○	×
PRINT# (USING)	×	○	○	○	○	○	×
POS	×	○	○	○	–	–	×
OPEN mode	I	O	O	I/O	I/O	I/O	I

NOTE:
○ or × in this table indicates that when a device is specified for a command or statement, the device

○: Can be used.
×: Cannot be used. An FC error occurs.
–: Causes no error but the command is invalid.

Refer to Chapter 3 for each command or statement.

1.7.1 Screen

If a file output device is specified as "SCRN:" (screen), it refers to either the LCD or external display which has been specified by a SCREEN command for text screen display. Characters to be output are actually written on the virtual screen and the screen that you can see at a time is limited by the display capacity of the LCD (or external display). Pay adequate attention to the output allocation and output speed.
The screen can also be used for programme debugging. When data is output on cassette or disk file, you will need a separate programme to check the data for correct output. In such a case, by changing the device name in the OPEN statement to "SCRN:", you can confirm visually the output data.

1.7.2 Printer

With the HX-20, you can use two types of printers. The first is the built-in microprinter which you must specify as "LPT0:". The other is an external printer connected to the HX-20 through the RS-232C interface, which you must specify as "COM0:".
As you may use the built-in microprinter more frequently, the following special output commands are provided in addition to the general output commands.

LPRINT, LPRINT USING
LLIST
KEY LLIST
COPY

By using a WIDTH<device name> command, you can also specify the print width. In this way, your design of output format can be greatly simplified.

1.7.3 Cassette

With BASIC, you can use two types of cassettes as auxiliary memory units.

"CAS1:"
Using a commercially available audio cassette, you can SAVE and LOAD programmes and data freely. When loading programmes with a LOAD command, or when a file is opened for input, specify the filename and the tape will be automatically searched for that file.

While executing a LOAD or OPEN command in the direct mode, the following message is displayed on the LCD screen each time a file other than the specified file is searched.

Skip: <filename>

When the specified file is found, the following message appears on the LCD screen.

Found: <filename>

Also, a LOAD? command may be used to skip the specified file. For further details, see LOAD? command in Chapter 3.

"CAS0:"
This device name indicates the optional microcassette drive. "CAS0" can be used in the same manner as "CAS1:". You can SAVE, LOAD, VERIFY or SEARCH the files using the built-in counter and executing a WIND command.

1.7.4 RS-232C port

Communications with external devices are regarded as most important in EPSON BASIC. For this reason, the RS-232C port is designed to allow programming in BASIC to set the conditions for communication not only with the external printer, but also with all other external devices. These conditions can be specified not only by an OPEN command, but also by the following commands.

LOAD
LIST
SAVE
RUN
MERGE

For further details, see Chapters 3 and 5.

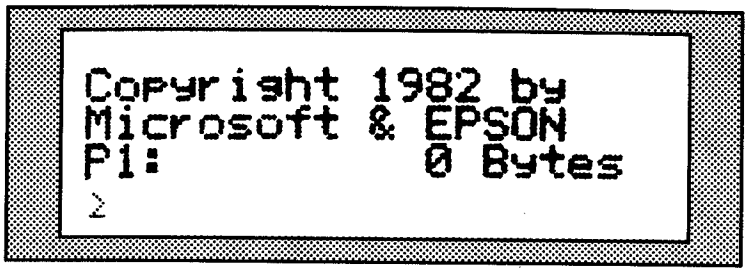
1.7.5 ROM cartridge (option)

You can use the optional ROM cartridge by specifying the device name as "PAC0:". Except that the ROM cartridge is used exclusively for input and that file loading can be accomplished in a much shorter time, it can be used in essentially the same manner as the microcassette drive. For these reasons, this device is extremely useful when you handle special programmes and/or data.

CHAPTER 2
Outline of EPSON BASIC

2.1 Operation modes

When BASIC is started up, your HX-20 displays the prompt sign ">" following the opening message. The prompt sign tells you that the HX-20 is now in command level, that is, an await state for command input.



If you input a programme statement written in accordance with the syntax established for BASIC but without line number, the statement is executed immediately upon pressing **RETURN** key. This type of programme execution is called "execution in the direct mode". Almost all BASIC commands and statements can be executed in the direct mode. The exceptions are those statements listed below.

INPUT
LINE INPUT
INPUT#
LINE INPUT#
RANDOMIZE

If you input a statement preceded by a line number (ranging from 0 to 63999), that statement and the line number are stored together in the memory as a program. The stored programme can then be executed by either a RUN command or a GOTO or GOSUB statement. This type of execution is called "execution in the program mode".
When a programme is executed by a RUN command, all variables (numeric, string and array variables and other defined statements) are cleared prior to execution. In contrast, a GOTO statement executes a programme without changing the current values of all variables.

2.2 Statements

A programme statement is a description of any of expressions, commands, statements, functions, etc., which is executed by BASIC. More than one statement can be specified by connecting them with a colon. These connected statements are called "multiple statements".

2.3 Lines

A programme line of BASIC always begins with a line number (represented by an integer in the range of 0 to 63999), followed by one or more statements and ends with a carriage return. A programme line can contain a maximum of 255 characters. Line numbers show the order in which the programme lines are stored in memory. Programme execution likewise follows this order starting from the lowest-numbered line. Line numbers are also used to access programmes for branching and editing. A full stop may be used instead of the line number after commands such as LIST and AUTO to instruct BASIC to operate on the last line, e.g., the line in which an error has occurred during programme execution or the last line input in the programme.

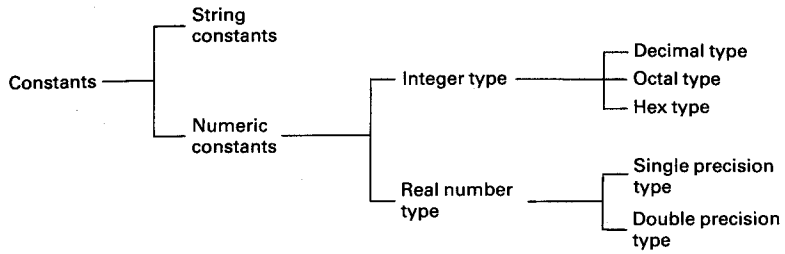
Examples: LIST.
AUTO.

2.4 Character set

The character set which can be used in BASIC consists of alphabetic (uppercase and lowercase) characters, numeric characters (0 to 9), special symbols and graphic characters. There are also a set of special control characters that perform the special functions as defined by their codes.
For details of these characters, see APPENDIX F, "Character Code Table" and APPENDIX G, "Control codes".
Also see section 2.10, DIP switch setting, and POKE command in Chapter 3 for character set selection.

2.5 Constants

Constants are values which are used by EPSON BASIC without any change for programme execution. The following types of constants are used in EPSON BASIC.



2.5.1 String constants

A string constant is a sequence of up to 255 alphanumeric characters and symbols enclosed in double quotation marks (").

Examples: "HX-20"
"Program-1"

Double quotation marks and control codes cannot be included in string constants. Use the CHR$ function to handle these marks and codes as characters.

2.5.2 Integer type numeric constants

(1) Decimal type
Whole numbers between −32768 to 32767 or numbers followed by the "%" sign. A decimal point cannot be suffixed to these numbers.

Examples: 12345
−9999
45678%

(2) Octal type
Octal numbers 0 to 7, prefixed by "&O" or "&".
Octal numbers are in the range of &0 to &177777.

Examples: &O123
&77777

(3) Hex type
Hexadecimal numbers 0 to F, prefixed by "&H". Hex numbers are in the range of &H0 to &HFFFF.

Examples: &H1F
&HABCD

2.5.3 Single precision type real numbers

Single precision constants are stored in 7 significant digits. Six of the 7 significant digits are displayed after rounding off the 7th digit.
A single precision constant is any numeric constant that has:
(1) seven or fewer digits, or
(2) exponential form using E, or
(3) a trailing exclamation mark (!).

Examples: 12345.6
−5E−38
234.5!

2.5.4 Double precision type real numbers

Double precision constants are stored in precision of 16 significant digits. Up to 16 significant digits are displayed.
A double precision constant is any numeric constant that has:
(1) eight or more digits, or
(2) exponential form using D, or
(3) a trailing number sign (#).

Examples: 3.141592653
−1.23 D 22
888.8#

2.6 Variables

Variables are a kind of location to take the values used in BASIC programmes. Each variable corresponds to a variable name consisting of alphanumeric characters.
If a numeric variable is referenced before a value is assigned to it, 0 is assigned as a numeric value. For a string variable, a null string, that is, a string of length 0, is assigned as a string value.

2.6.1 Variable names and declaration characters

(1) A variable name is represented by a sequence of up to 255 characters beginning with an alphabetic character. Only the first 16 characters of a name are identified.
A variable name can contain a reserved word but cannot begin with a reserved word. (Reserved words are keywords such as commands, statements and functions.)
Variable names starting with alphabetic characters "FN" are not allowed.
In variable names, capitals and lowercase letters are regarded as the same.

(2) Variable types are determined by type declaration characters. A declaration character written as the last character of a variable name declares the type of the variable. If a declaration character is omitted, the variable type is assumed as a single precision type real number.
(3) BASIC distinguishes different variable types under the same variable name.

Type declaration characters

- % Integer variable
- ! Single precision variable
- # Double precision variable
- $ String variable

2.6.2 Array variables

An array refers to a variable whose several elements can be referenced by one variable name. Each element in the array is referenced by an "array variable" that is subscripted by integer notation. The dimensions of an array variable and the maximum value of a subscript depend on the capacity of the memory. (See DIM in Chapter 3.)

2.7 Type conversion

When necessary, EPSON BASIC automatically converts a numeric constant from one type to another.

(1) If you assign a numeric data of one type to a numeric variable of another type, the numeric constant is converted to and stored as the type declared in the variable name. If you attempt to assign a numeric variable to a string variable or vice versa, a "Type mismatch" error occurs.
Example: A%=32.34
In this example, 32 will be stored in A%.

(2) During expression evaluation, all of the operands in an arithmetic or relational operation are converted, and their results are returned to the same degree of precision of the most precise operand.

Examples: 10#/3
10#/3#

The above two expressions are evaluated at the same degree of precision. However, if you neglect to pay attention to the number of significant digits in the variable and to the order of executing operations, an error may result.

A#=10/3*3#
B#=10#/3*3

In the above example, 9.999999761581421 will be stored in variable A# and 1.0E+01 will be stored in B#.

(3) In logical operations, all the logical operands are converted to integers and an integer result is returned.
An "Overflow" error will occur if the operands are not in the range −32768 to 32767 when converted to integers.

Example: A=NOT 123.456
In this example, −124 will be stored in A.

(4) When a real number is converted to an integer, the fractional portion is truncated. If the real number when converted to an integer is not in the range −32768 to 32767, an error also occurs.

Example: A%=777.7
778 will be stored in A.

(5) If a double precision variable is assigned to a single precision variable, only the first seven digits rounded down will be stored.

Example: P!=3.141592653589793
3.141593 will be stored in P! and
displayed as 3.14159

2.8 Expressions and operations

An expression refers to ① a string or numeric constant, ② a string or numeric variable, and ③ a combination of string or numeric constants and variables connected by operators.

Examples: "ABCD"
1.41421356
2*3
A+B/C

BASIC employs the following five types of operators in performing mathematical or logical operations.

(1) Arithmetic operators
(2) Relational operators
(3) Logical operators
(4) Functional operators
(5) String operators

2.8.1 Arithmetic operators

(1) EPSON BASIC uses the following arithmetic operators

Operator	Operation	Sample expression
^	Exponentiation	A^B
−	Negation	−A
*, /	Multiplication and division of real numbers	A*B, A/B
+, −	Addition and subtraction	A+B, A−B

Parentheses are used to change the order of operations. Operations within parentheses are performed first.
Shown below are the representations of the algebraic expressions in BASIC.

EPSON BASIC	Algebraic expression
3*X+Y	3X+Y
X/Y−Z	(X÷Y)−Z
X^2+Y*3+4	X^2+3Y+4
Y^Y^Z	$(X^Y)^Z$
X*(−Y)	X(−Y)

(2) Integer division and modulus arithmetic
Integer division is denoted by the backslash (\). If operands are real numbers, they are converted to integers before execution. Quotients are truncated to integers.
Example: A%=10\3
B%=9.5\3.3
The value stored in both A% and B% will be 3.
Modulus arithmetic, denoted by the operator MOD, returns the integer value of the remainder of an integer division.

Example: A%=10 MOD 3
1 will be stored in A%.

(3) Division by zero and overflow
If a division by zero occurs during the evaluation of an expression, a "Division by zero" error occurs and execution of the operation is terminated. If the result of an evaluation or a value assignment exceeds the number of digits that can be handled by the variable, an "Overflow" occurs and execution stops.

2.8.2 Relational operators

Relational operators are used to compare two values. The result of this comparison is either "true" (−1) or "false" (0). This result is then used to make a decision regarding programme flow.

Operator	Relation tested	Sample expression
=	Equality	X=Y
< >, ><	Inequality	X<>Y, X><Y
<	Less than	X<Y
>	Greater than	X>Y
<=, =<	Less than or equal to	X<=Y, X=>Y
>=, =>	Greater than or equal to	X<=Y, X=>Y

Example: X=A=0
If A=0, −1 will be stored in X, and
if A<>0, 0 will be stored in X.

2.8.3 Logical operators

Logical operators are used to perform tests on multiple relations, bit manipulation, or Boolean operations. Logical operators return a result for each bit that is either a true (1) or false (0).

NOT (Negation)	
X	NOT X
1	0
0	1

AND (Logical product)		
X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR (Logical sum)		
X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR (Exclusive-OR)		
X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0
IMP (Implication)		
X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1
EQV (Equivalence)		
X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

Logical operators can be used to connect two or more relational operators to make decisions on complex conditions with a single statement. If the value given by the logical operator is −1 (true) or 0 (false), the result obtained must be either −1 or 0.

Example: IF A<0 AND B=0 then 100
In this example, if A is negative and B is 0, program control branches to line 100.

Logical operators convert their operands into two's complement integers in the range −32768 to 32767 before any operations. If the operands are not in this range, an error occurs. The given operation is performed bit by bit on these integers.

Example: A=21 AND 13
$21=(10101)_2$, $13=(01101)_2$
$(10101)_2$ AND $(01101)_2=(00101)_2$
$(00101)_2=5$
Thus the value entered in A is 5.

2.8.4 Functional operators

A function is used in an expression to call a predetermined operation that is to be performed on a given argument.
BASIC has "Intrinsic Functions" consisting of numeric functions such as SIN, SQR, etc., and string functions such as RIGHT$, STR$, etc. For details of these functions, please refer to Chapter 4.

As explained later at DEFFN in Chapter 3, BASIC also allows "User defined" functions. If a real number is assigned to a function which normally takes only an integer as its argument, the fractional portion of the real number is rounded off to the nearest integer and then the functional operation is performed. A double precision real number may be assigned as the argument of a numeric function. In this case, however, the operation is performed in single precision.

2.8.5 String operations

Strings may be concatenated using a plus sign (+).

Example: A$="ABC"+CHR$(34)
ABC" will be stored in A$.

Comparison of strings can also be made using relational operators.

=, <, >, <>, ><, <=, =<, >=, =>

Strings are compared by taking one character at a time from each string. If two strings compared are the same, both the strings are judged "equal". If any of the codes differs, the lower character code number precedes the higher. In string comparison, the shorter string is considered smaller. Leading and trailing blanks are significant in string comparison.

Example: "AAA"="AAA"
"X" < "Y"
"XYZ" > "XY"
"ABC" > "A BC"

2.8.6 Precedence order of operations

Mathematical and logical operations are performed in the following order.

1. Operations within parentheses
2. Functions
3. Exponentiation (^)
4. Negation (−)
5. Multiplication and division of real numbers (*, /)
6. Integer division (\)
7. Modulus integer division (MOD)
8. Addition and subtraction (+, −)
9. Relational operands
10. NOT
11. AND
12. OR
13. XOR
14. IMP
15. EQV

2.9 Error messages

If EPSON BASIC detects an error which causes the execution of a programme to stop, an error message is printed and the HX-20 returns to command level.
The format for error messages in the direct mode is:

XX Error

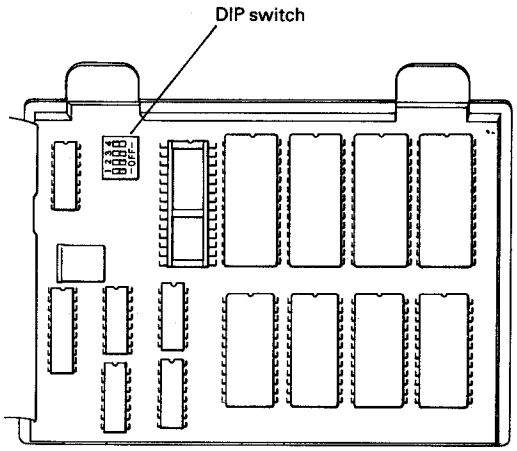
The format in the programme mode is:

XX Error in nnnn

XX is the error code and nnnn is the line number where the error was detected. For details of the BASIC error messages, refer to APPENDIX A.

2.10 DIP switch setting

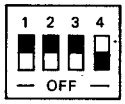
The DIP switch setting can be changed by opening the cover on the back of the HX-20 as shown below.



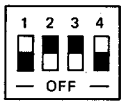
Location of the DIP Switch

The DIP switch settings for the character codes for each country are as shown below.

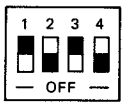
USASCII



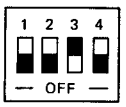
France



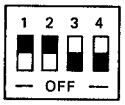
Germany



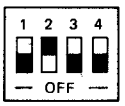
England



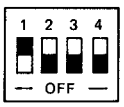
Denmark



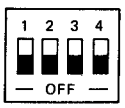
Sweden



Italy



Spain



You can also do this by software. (See POKE in Chapter 3.)
For the respective countries' character codes, see APPENDIX F.

How to use Chapters 3 and 4

All the commands and statements of EPSON BASIC are explained in alphabetical order for easy reference in Chapters 3 and 4. All I/O commands related to peripheral devices are explained collectively.
For details of the operations and features peculiar to the respective peripherals, refer to Chapters 1 and 5.
Commands and statements prefixed with an asterisk "*" are those unique to BASIC. As these commands and statements operate differently from conventional BASIC commands and statements, please read the explanation of each of these commands with great care.
The descriptions of BASIC commands and statements are presented in the following format.

FORMAT Explains how to write the command or statement. Follow the syntax described below when you input each command or statement.

1. Input items shown in uppercase letters using either uppercase or lowercase letters. Make this distinction, particularly for those items enclosed in double quotation marks (filenames, etc.).
2. Items surrounded by "< >" are those to be specified by the user.
3. Items enclosed in brackets "[ ]" are optional and can be omitted. If you omit these items, the HX-20 will supply the default values or previously specified values.
4. Symbols other than those described above, e.g., parentheses, commas, colons, semicolons, hyphens, equal signs, etc., must be input exactly at the positions as they are shown in the manual.
5. Optional items indicated by ". . ." can be repeated as many times as desired within a maximum of 255 characters.
   Example: <variable>[,<variable>. . .]
   A, B$, C!, D, etc., may be repeated.
6. When two or more items appear between two vertical lines, this means that you must specify the desired function by selecting one of the items for input.

Example:

THEN	**<statement>**
	<line number>
GOTO	**<line number>**

In this example, the possible choices for input are:

- **THEN <statement>**
- **THEN <line number>**
- **GOTO <line number>**

7. As a rule, BASIC ignores spaces. However, spaces are not permitted within variable names and keywords. If a variable is to be followed by a keyword, a space must be inserted to delimit them for correct operation.

PURPOSE Describes briefly how the command or statement functions.
EXAMPLE This shows a simple example of actual input.
REMARKS Hints on the correct use of each command or statement are given, along with a description of their functions.
SAMPLE PROGRAMME Sample programmes using the command or statement are presented for exercise.

In the HX-20, 5 programmes are independently managed in the respective programme areas. Therefore, to avoid the accidental destruction or loss of important data or programmes in the memory, be sure to check the programme area using a STAT or LIST command, before modifying any programme.

CHAPTER 3
Commands and Statements

CLEAR

FORMAT **CLEAR [<character area size>[,<RAM file size>]]**

PURPOSE To initialise variables and to set the size of the character area and the RAM file.

EXAMPLE **CLEAR 200, 256**

REMARKS CLEAR command sets all numeric variables to 0 and all string variables to null. CLEAR also closes all OPENed files and voids all data defined by DEF statements (DEFFIL, DEF FN, DEF USR, DEFINT, etc.). <character area size> is the size, in bytes, of the memory area used by BASIC for character string processing.

<character area size> is set to 200 bytes at each warm start. When executing operations on a large number of character strings, or when using a large character array, specify sufficiently large <character area size>, or an "OS" error will occur.
<RAM file size> sets, in bytes, the area of the memory to be used for RAM file storage. <RAM file size> is set to 256 bytes at each cold start and is not affected by warm start.

When using a CLEAR command to set <RAM file size>, <character area size> cannot be omitted. Also note that when changing the size of a RAM file in which data is stored, if the newly set size is smaller than the previously set size, data in excess of the new size will become void.

SAMPLE PROGRAMME

```
LIST

100 FOR I=1 TO 10
110 A(I)=I : NEXT I
120 GOSUB 200
130 CLEAR
140 GOSUB 200
150 END
200 FOR I=1 TO 10
210 PRINT USING"####";A(I);
220 NEXT I
230 PRINT
240 RETURN
RUN
   1   2   3   4   5   6   7   8   9  10

   0   0   0   0   0   0   0   0   0   0

>
```

CLOSE

FORMAT **CLOSE[[#]<file number>,[#]<file number>...]]**
PURPOSE To close file(s).
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 programme is edited, all the files being open at that time will be closed.

(See END, OPEN and 5.2, Sequential Files.)

*CLS

FORMAT **CLS**
PURPOSE To clear a text screen.
EXAMPLE **CLS**

REMARKS CLS clears only the text screen (virtual screen) on the LCD or external display and returns the cursor to its home position (i.e., the upper left-hand corner of the virtual screen). However, when a CLS is executed with both text and graphic screens being displayed on the LCD, the CLS will also clear the graphic screen. CLS functions the same as PRINT CHR$ (12).

(See GCLS.)

COLOR

FORMAT **COLOR [<foreground colour>][,[<background colour>][,<colour set>]]**

PURPOSE To specify the screen colours of the external display.

EXAMPLE **COLOR 0, 3, 0**

REMARKS COLOR is used to set the foreground colour (i.e., display colour of lines and characters) and background colour of the external display. This command is effective only when the HX-20 is connected to an external display (e.g., TV) through the optional display controller.

You can specify colours using colour codes 0 to 3. The foreground colour and the background colour can be specified independently. A total of 8 colours are available for display. A maximum of 4 colours can be output simultaneously and either of two colour sets can be selected using code "0" or "1". The colour codes and corresponding colours are shown below.

	Colour code
Colour set 0	0: Green
	1: Yellow
	2: Blue
	3: Red
Colour set 1	0: White
	1: Cyan
	2: Magenta
	3: Orange

The default values at warm start are as follows.
<Foreground colour>: 1
<Background colour>: 0
<Colour set> : 0

(See SCREEN.)

CONT

FORMAT CONT
PURPOSE To resume the execution of a programme that has been stopped.
EXAMPLE CONT

REMARKS If you press **BREAK** key during programme execution, or when a STOP in a programme is executed, execution of the programme will stop and the message "Break in XXXX" will be displayed. At this point, the programme execution resumes upon input of a CONT command.
CONT is usually used together with a STOP statement for programme debugging.
If you press **BREAK** key during the I/O operation to a printer or a cassette, message "Abort" will be displayed. In this case, programme execution cannot be resumed by a CONT command. This command is also invalid if the programme is edited after it was stopped by **BREAK** key or STOP.

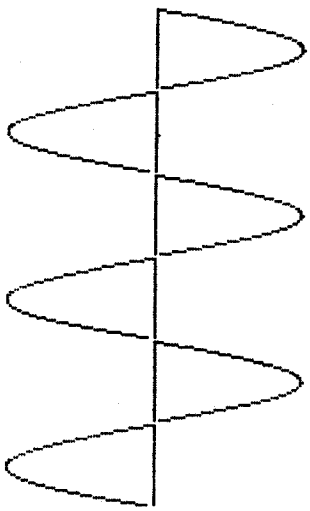
COPY

FORMAT COPY
PURPOSE To output the characters and graphics displayed on the LCD, on the built-in microprinter.
EXAMPLE COPY

REMARKS COPY command, unlike an LPRINT statement, provides hard copy without any space between lines. Therefore, with this command, you can print the graphics drawn using LINE and PSET statements exactly as they are displayed on the screen.

SAMPLE PROGRAMME

```
100 PI=3.141592
110 DX=PI/64
120 CLS
130 PRINTCHR$(23)
140 LINE(60,31)-(60,0),P
SET
150 FOR Y=0 TO 63
160 A=SIN(X)*59
170 X=X+DX
180 LINE-(A+60,Y/2),PSET
190 NEXT Y
200 COPY
210 GOTO 130
220 END
```



DATA

FORMAT DATA <constant>[,<constant>...]

PURPOSE To store the numeric and string constants that are accessed by the READ statement(s).

EXAMPLE DATA HX, 20, EPSON

REMARKS DATA statements are nonexecutable and may be placed anywhere in the programme and any number of DATA statements may be used in a programme.

Any type of constant can be used in <constant> (e.g., single precision, double precision, integer), while no numeric expressions (e.g., 3*4) are allowed. The variable type (numeric or string) given in the READ statement must agree with the corresponding constant in the DATA statement.

Each <constant> in a DATA statement must be separated by commas. String constants must be surrounded by double quotation marks if they contain commas, colons or significant leading or trailing spaces. Otherwise, quotation marks are not needed.

(See READ and RESTORE.)

SAMPLE PROGRAMME

```
10 READ A,B,C
20 PRINT A;B;C
30 DATA 1,2,3,4,5,6,7,8,
9,10
40 READ A,B,C
50 PRINT A;B;C
60 READ A,B,C
70 PRINT A;B;C
80 END
```

```
 1  2  3
 4  5  6
 7  8  9
```

DEFFIL

FORMAT DEFFIL <record length>, <relative address>

PURPOSE To define the relative address of record 0 in a RAM file and the length of a single record.

EXAMPLE DEFFIL 20, 200

REMARKS <relative address> is the number of bytes from the starting address of the RAM file area specified by a CLEAR statement. <record length> defines the length, in bytes, of a single record in the RAM file area to be used. When BASIC is warm-started, <record length> is set at 255 bytes and <relative address> at 0. Execution of a CLEAR statement will also void the <record length> and <relative address> previously defined by the DEFFIL statement and the default values become 255 and 0 respectively at the time of a warm start.

(See GET%, PUT% and 5.1, RAM Files.)

SAMPLE PROGRAMME

```
LIST

100 DEFINT I,J,K
110 DEFFIL 2,0
120 FOR I=0 TO 15
130 PUT% I,I
140 PRINT USING "  & &";HEX$(I);
150 NEXT I:PRINT
160 DEFFIL 2,1
170 FOR J=0 TO 15
180 GET% J,K
190 PRINT USING "  & &";HEX$(K);
200 NEXT J
210 END
RUN
  0    1    2    3    4    5    6    7
  8    9    A    B    C    D    E    F

  0    100  200  300  400  500  600  700
  800  900  A00  B00  C00  D00  E00  F00
>
```

DEF FN

FORMAT DEF FN<name>[(<parameter>[,<parameter>....])]=
<function definition>

PURPOSE To define a function created by the user.

EXAMPLE DEF FNZ(X, Y)=X*2+Y*3+A

REMARKS <name> is the name of a function and must begin with alphabetic characters except reserved words (and is subject to the same restriction as variable names). The list of parameters comprises those variable names in the function definition that are to be replaced when the function is called. The items in the list are separated by commas. <function definition> is an expression that performs the operation of the function. It is limited to one line.

Variable names that appear in this expression serve only to define the function; they do not affect programme variables that have the same name. If a variable name used in <function definition> does not appear in the argument list, the current value of the variable is used.

User-defined functions may be numeric, string or a combination of both. However, the type of the defined argument must match that of the variable actually called. A DEF FN statement must be executed to define a function before it is called.

If a function is called before it is defined, a UF (undefined user function) error will occur.

SAMPLE PROGRAMME

```
100 DEFFNA(B,C,R)=SQR(B^
2+C^2-2*B*C*COS(R/180*3.
1415926))
110 INPUT"SIDE B=";B
120 INPUT"SIDE C=";C
130 INPUT"ANGLE R=";R
140 A=FNA(B,C,R)
150 PRINT "SIDE A= ";A
160 GOTO 110
170 END
```

DEFINT/SNG/DBL/STR

FORMAT DEF|INT|SNG|DBL|STR| <range(s) of letters>

PURPOSE To declare variable types as integer, single precision, double precision, and string.

EXAMPLE DEFSTR A,X–Z

REMARKS DEFINT, DEFSNG, DEFDBL and DEFSTR declare that the variable names beginning with the letter(s) specified by <range(s) of letters> will be integer type, single precision type, double precision type and string type, respectively.

To declare a variable or a range of variables, a single letter or two letters linked with a hyphen will be used as the letter(s) to be specified by <range(s) of letters>. To declare two or more ranges of variables, the letter(s) specifying each range of letters will be separated by a comma. However, a type declaration character always takes precedence over a DEF type statement in the typing of a variable. If no type declaration statements are encountered, BASIC assumes that all variables without declaration characters are single precision variables.

SAMPLE PROGRAMME

```
LIST

100 DEFINT A-C
110 DEFSNG E-G
120 DEFDBL I-K
130 A =123.456789
140 B#=123.456789
150 D =123.456789
160 E =789.123456
170 F%=789.123456
180 H#=789.123456
190 I =0.98765432
200 J%=0.98765432
210 L =0.98765432
220 PRINT A ,B#,D
230 PRINT E ,F%,H#
240 PRINT I ,J%,L
250 END
RUN
 123            123.456789     123.457
 789.124        789            789.123456

 .98765432      1              .987654
>
```

DEF USR

FORMAT DEF USR[<digit>]=<starting address>
PURPOSE To specify the starting address of a machine language subroutine.
EXAMPLE DEF USR6=&H0C00

REMARKS DEF USR specifies the starting address of the machine language routine called by the USR function. <digit> may be any digit from 0 to 9. A total of 10 USR functions are permitted. If <digit> is omitted, 0 is assumed.

(See USR and 5.3, Machine Language Programmes.)

SAMPLE PROGRAMME

```
MEMSET &H0A41

LIST

100 POKE &H0A40:&H39
110 DEFUSR0=&H0A40
120 ' &H39 = RETURN
130 A=USR(0)
140 PRINT A
150 END
RUN
 0
>
```

DELETE

FORMAT DELETE [<starting line number>][–[<ending line number>]]
PURPOSE To delete specified programme lines.
EXAMPLE DELETE 100–200
DELETE 100–
DELETE –200
DELETE 100

REMARKS DELETE command deletes all or part of the programme currently LOGged IN as specified by <starting line number> and <ending line number>. If only <starting line number> is specified, only that programme line is deleted. <starting line number> followed by a hyphen deletes that line and all higher-numbered lines. If <ending line number> preceded by a hyphen is specified, all the lines from the beginning of the programme through that line are deleted. When the DELETE command is executed, all files OPENed at the time of execution are closed.

(See LOGIN.)

SAMPLE PROGRAMME

```
100 'DELETE ELIMINATES
110 'UNEEDED LINES
120 'DELETE LINES 120 TO
 140
130 '****UNEEDED LINE***
140 '****EXCESS LINE***
150 '##IMPORTANT LINE##
160 '##NECESSARY LINE##

DELETE 120-140

100 'DELETE ELIMINATES
110 'UNEEDED LINES
150 '##IMPORTANT LINE##
160 '##NECESSARY LINE##
Ok
```

DIM

FORMAT DIM<variable>(<maximum subscript value>[,<maximum subscript value>...])[,...]
PURPOSE To declare the size of array variable elements.
EXAMPLE DIM A(40, 10), B$(50)

REMARKS DIM specifies the maximum value for array variable subscripts and allocates storage accordingly.

If an array variable name is used without a DIM statement, the maximum value of its subscript(s) is assumed to be 10. The minimum value for a subscript is always 0, unless specified as 1 with the OPTION BASE statement. If a subscript that is greater than the specified maximum is used, a BS ("Bad subscript") error occurs. The DIM statement sets all the elements of the specified numeric array to an initial value of zero, and those of the specified character array to null.

(See ERASE and OPTION BASE.)

END

FORMAT END
PURPOSE To close all files and terminate programme execution.
EXAMPLE END

REMARKS END terminates programme execution after closing all files and then returns BASIC to command level.
An END statement can appear anywhere and as often as required in a program. The END statement is optional at the end of a programme. In this case, no files will be closed.

ERASE

FORMAT ERASE <array variable>[,<array variable>...]
PURPOSE To eliminate arrays from a programme.
EXAMPLE ERASE A, B

REMARKS ERASE eliminates the previously dimensioned arrays specified by the list of array variables. After arrays have been erased, they can be redimensioned using a DIM statement. If an attempt is made to redimension an array without first erasing it, a DD ("Duplicate definition") error occurs.

(See DIM.)

SAMPLE PROGRAMME

```
LIST

100 OPTION BASE 1
110 D=10
120 FOR I=1 TO 10
130 A(I)=11-I
140 NEXT I
150 GOSUB 300
160 ERASE A
170 D=20
180 DIM A(20)
190 FOR I=1 TO 20
200 A(I)=I
210 NEXT I
220 GOSUB 300
230 END
300 FOR I=1 TO D
310 PRINT USING"####";A(I);
320 NEXT I
330 PRINT
340 RETURN
RUN
  10   9   8   7   6   5   4   3   2   1

   1   2   3   4   5   6   7   8   9  10
  11  12  13  14  15  16  17  18  19  20

>
```

ERROR

FORMAT ERROR<integer expression>

PURPOSE To simulate the occurrence of an error; or to allow error codes to be defined by the user.

EXAMPLE ERROR 225

REMARKS When an ERROR statement is executed without ON ERROR GOTO, the message of the error code corresponding to the <integer expression> will be output and the programme execution will stop.
If an ERROR statement specifies a code for which no error message has been defined, the HX-20 will respond with a message "UP Error". In all cases the value of the error code will be assigned to variable ERR and the line number in which the ERROR statement was encountered will be assigned to variable ERL.
An unused error code can be defined by the user with an ERROR statement, and this code can in turn be used in an error processing programme with an ON ERROR GOTO statement. <integer expression> must be in the range 1 to 255.

(See ON ERROR GOTO, RESUME, ERL/ERR, and APPENDIX A, Error Messages.)

SAMPLE PROGRAMME

```
LIST

100 INPUT "ERROR NUMBER"
;A
110 ERROR A
120 END

RUN
ERROR NUMBER? 78
UP Error In 110

RUN
ERROR NUMBER? 4
OD Error In 110

RUN
ERROR NUMBER? 24
WE Error In 110
2
```

EXEC

FORMAT **EXEC [<starting address>]**
PURPOSE To start the execution of a machine language subroutine.
EXAMPLE **EXEC &H0C00**

REMARKS EXEC executes a machine language subroutine from the <starting address>. However, before execution of an EXEC command, the machine language subroutine must be loaded into the memory by executing a LOADM command or similar statement.

When EXEC is used within a programme, the BASIC programme execution continues with the BASIC statement immediately following the EXEC statement after the machine language subroutine has terminated.
When <starting address> is not specified, execution begins from the <starting address> of the LOADM command or the EXEC command previously executed. BASIC command are ineffective while the machine language subroutine is being executed. The BASIC programme and the RAM file are not protected against careless use by the machine language subroutine and ample precautions must be taken to avoid their destruction.

(See LOADM, USR and 5.3 Machine Language Programmes.)

FILES

FORMAT FILES["<device name>"]
PURPOSE To display the names of all files residing on a specified memory.
EXAMPLE FILES "CAS1:"

REMARKS The following information is displayed for all files residing in the auxiliary memory specified by <device name>: filename, filetype, classification, recording format. Classification is displayed as a single digit,

0: BASIC programme file
1: Data file
2: Machine language programme file

Recording format is shown with either "A" or "B"

A: ASCII format
B: Binary format

If <device name> is omitted, all of the peripheral devices currently connected are automatically searched and the device with the highest precedence (i.e., default device) is assumed. If no peripheral device is connected, "CAS1:" is assumed. In BASIC, there is a feature for the detection of the end of files, but none for the detection of the end of tapes. For this reason, when "CAS1:" or "CAS0:" is specified as the <device name>, the cassette tape will run indefinitely. You must determine the end of the file, and press the **BREAK** key at the appropriate moment to stop execution.

FOR...TO...STEP–NEXT

FORMAT **FOR <variable>=<initial value> TO <final value>**
[STEP <increment>]
.
.
.
NEXT[<variable>[,<variable>...]]

PURPOSE To allow a series of instructions between FOR and NEXT statements to be performed in a loop a given number of times.

EXAMPLE
```
FOR I=0 TO 100 STEP 5
.
.
.
NEXT I
```

REMARKS <variable> is used as a counter. The counter is first set to the value specified by <initial value>. BASIC executes the programme lines following the FOR statement until the NEXT statement is encountered. Then the counter is incremented by the <increment> amount specified by STEP and a check is performed to see if the value of the counter is greater than <final value>. If it is not greater, BASIC returns to the programme line following the FOR statement to repeat the same processing. If it is greater, execution continues with the statement following the NEXT statement.

Numeric expressions may be used for <initial value>, <final value> and <increment>. If STEP is not specified, <increment> is assumed to be one. If <increment> is negative, the <final value> of the counter is set to be less than the <initial value>.

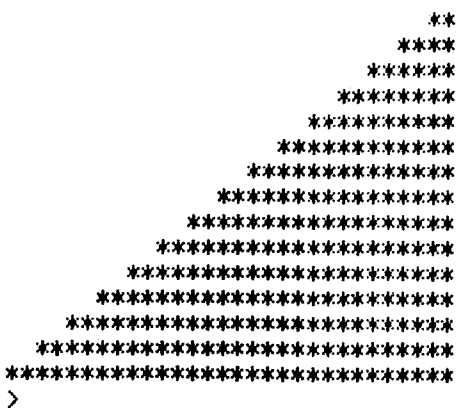
If <increment> is positive and the <initial value> of the counter is greater than the <final value>, or if <increment> is negative and the <initial value> of the counter is not greater than the <final value>, no FOR...NEXT loop will be executed. However, the <initial value> will be assigned to the <variable>.
FOR...NEXT loops may be nested, that is, a FOR...NEXT loop may be placed within the context of another FOR...NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop.
If a NEXT statement is encountered before its corresoponding FOR statement, a "NEXT without FOR" error message is issued and programme execution is terminated.

If nested loops have the same end point, a single NEXT statement may be used for all of them, by separating the variables to be represented with commas (e.g., NEXT I,J,K).

SAMPLE PROGRAMME

```
LIST

100 FOR I=30 TO 0 STEP -2
110 FOR J=1 TO I
120 PRINT" ";
130 NEXT J
140 FOR K=1 TO 30-I
150 PRINT"*";
160 NEXT K
170 PRINT
180 NEXT I
190 END
RUN
```



```
>
```

GCLS

FORMAT GCLS
PURPOSE To clear a graphic screen.
EXAMPLE GCLS

REMARKS GCLS clears only the graphic screen being displayed on the LCD or external display. If both text and graphic screens are displayed on the LCD, the text screen will be left as is.

(See CLS.)

GET%

FORMAT GET% <record number>,<variable name>[,<variable name>...]
PURPOSE To read data from a RAM file into variables.
EXAMPLE GET% O, A!, B#, C$

REMARKS GET% statement reads data from the record in the RAM file specified by <record number> into the variables specified by the list of variable names. <record numbers> start from 0.
Before executing a GET% statement, the length of a single record and the position of the file containing that record must be defined by a DEFFIL statement.
The types of the variable names in the GET% statement must correspond one-to-one with the types of the variable names in the PUT% statement. Also, when using string variable names, they should be added at the end of the variable list. When GET% and PUT% are to be used with multiple DEFFIL statements, careful attention must be paid to the length and record number of each record, as well as to the method of storing variable values.

(See DEFFIL, PUT%, and 5.1, RAM Files.)

GOSUB...RETURN

FORMAT GOSUB <line number>
.
.
.
RETURN

PURPOSE To branch to and return from a subroutine.

EXAMPLE GOSUB 500

REMARKS GOSUB causes an unconditional break in programme execution by transferring control to a subroutine specified by <line number> which is the first line of the subroutine. Programme control returns to the line following the most recent GOSUB statement when a RETURN statement in the subroutine is executed.
Subroutines are the independent portions of a main programme each of which contains a RETURN statement at the end of the programme.
Any of these subroutines may be called any number of times in a programme by a GOSUB statement. A subroutine may also be called from within another subroutine. Such nesting of subroutines is limited only by available memory.
A subroutine may contain more than one RETURN statement, if logic dictates a return at different points in the subroutine and if each RETURN statement corresponds correctly to a GOSUB.

SAMPLE PROGRAMME

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 Z=X
40 GOSUB 200
50 X1=ANS
60 Z=Y
70 GOSUB 200
80 X2=ANS
90 PRINT "X=";X,"Y=";Y
100 PRINT "(2*X+3)^2+(2*
Y+3)^2"
110 PRINT "=";X1+X2
120 END
200 ANS=(2*Z+3)^2
210 RETURN
```

GO TO/GOTO

FORMAT (1) **GO TO <line number>**, or
(2) **GOTO <line number>**

PURPOSE To branch programme execution to a specified line number.

EXAMPLE **GOTO 300**

REMARKS When a GOTO statement is executed, programme control branches unconditionally to a specified line number.
Formats (1) and (2) provide exactly the same function.

IF...THEN...ELSE/IF...GOTO...ELSE

FORMAT IF <logical expression>

```
| THEN |<statement>|  | [ELSE |<statement>| ]
|      |<line No.> |  |       |<line No.> |
| GOTO <line No.>  |
```

PURPOSE To choose a particular route for programme execution based on conditions established in a logical expression.

EXAMPLE IF A >10 THEN A=0 ELSE 200

REMARKS IF controls programme execution according to the conditions established in <logical expression>. If the result of <logical expression> is not zero (i.e., true), the THEN or GOTO statement is executed. If the ELSE statement is omitted, the execution continues with the next executable statement. Within the IF . . . THEN . . . ELSE statement, separate IF statements may be nested to create a multiplexed statement, limited to one line.

SAMPLE PROGRAMME

```
100 DEFSTR A-Z
110 DEF FNS=RIGHT$(TIME$
,1)
120 DEF FND=MID$(TIME$,7
,1)
130 CLS
140 OD=FND
150 OS=FNS
160 IF OS=FNS THEN 160
170 LOCATE 0,0,0
180 PRINT TIME$
190 IF OD<>FND THEN SOUN
D 12,4 ELSE SOUND 19,1
200 GOTO 140
210 END
```

INPUT

FORMAT INPUT ["<prompt string>"] | ; | , | <variable>[,<variable>...]

PURPOSE To allow input from the keyboard into a specified variable during programme execution.

EXAMPLE INPUT"NAME"; A$

REMARKS When an INPUT statement is encountered, execution of the programme pauses and the HX-20 waits for input from the keyboard. If <prompt string> is followed by a semicolon, the <prompt string> is displayed on the LCD followed by a question mark "?" with a significant 1-digit space.
When <prompt string> is followed by a comma, it appears on the LCD by itself with no following punctuation.
Data is input by pressing **RETURN** key and is assigned to a specified variable. When multiple variables, separated by commas, are specified, the data must also be separated by commas for input and must match the variables in terms of number and type. If they do not match, the system will display a "?Redo" message and return to a wait state for data input.
String constants need not be enclosed in double quotation marks for input unless they contain commas, colons or significant spaces.
INPUT is invalid in the direct mode.

(See LINE INPUT.)

SAMPLE PROGRAMME

```
100 INPUT"STRING";A$
110 INPUT"NUMBER";A
120 PRINT"THE STRING AND
 THE NUMBER ARE ";A$A

STRING? HX
NUMBER? 20

>
THE STRING AND THE NUMBE
R ARE HX 20
```

INPUT#

FORMAT **INPUT# <file number>, <variable>[,<variable>...]**
PURPOSE To read data items from a specified file and assign them to programme variables.
EXAMPLE **INPUT#1, A, B, C$**

REMARKS Other than the facts that data is read from a specified file and that no question mark is output, INPUT# functions essentially the same as an INPUT statement.
<file number> must be the number used when the file was OPENed for input. The data items in the file should be as those required in an INPUT statement. When you use an INPUT# statement to read data from a data file, data must be already prepared in the file. If the execution of the INPUT# statement continues after all the data in the file has been read, an IE ("Input past end") error occurs. If you observe this point, all the data which has been written into the data file using several PRINT# statements can be read with a single INPUT# statement.

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

KEY

FORMAT **KEY <key number>, <string>**
PURPOSE To define the programmable function keys.
EXAMPLE **KEY 1,"LIST"**

REMARKS The keyboard of the HX-20 is equipped with 5 "programmable function keys" so that each key may be assigned to the function described in any string. With these keys combined with the SHIFT mode, a total of 10 strings can be defined. The length of character string including a control code must be a maximum of 15 characters. Characters that cannot be input from the keyboard can be specified using the function CHR$ appended to the string with a plus (+) sign.

(See KEY LIST.)

KEY LIST/KEY LLIST

FORMAT (1) **KEY LIST**
(2) **KEY LLIST**

PURPOSE To output the strings assigned to the programmable function keys on the screen and the microprinter, respectively.

EXAMPLE **KEY LLIST**

REMARKS (1) KEY LIST causes a complete list of strings assigned to the function keys to be displayed along with the key numbers. Each control code is displayed by typing the upper arrow "^" and a letter.
(2) KEY LLIST is the same as KEY LIST except that it causes a complete list of strings to be output on the built-in microprinter. When BASIC is started, the following strings are assigned to the respective function keys.

PF1	AUTO	PF6	?DATE$:?TIME$^M
PF2	LIST^M	PF7	LOAD
PF3	LLIST^M	PF8	SAVE
PF4	STAT	PF9	TITLE
PF5	RUN^M	PF10	LOGIN

LET

FORMAT **[LET]<variable>=<expression>**

PURPOSE To assign the value of an expression to a variable.

EXAMPLE **LET A=3.141592**

REMARKS LET assigns a value or the value of <expression> to <variable>. The actual assignment is performed by the equal "=" sign and LET can be omitted. <expression> may be a numeric or string constant. However, assignment of a string variable to a numeric variable and the reverse are not permitted. When assigning unmatched types of numeric variables, the variable type is to the right of the equal sign is converted into the one to the left of the equal sign before the assignment is performed.

LINE

FORMAT LINE[(<horizontal coordinate 1>, <vertical coordinate 1>)]–(<horizontal coordinate 2>, <vertical coordinate 2>), |PSET / PRESET| [,<colour>]

PURPOSE To draw a straight line between two specified points.

EXAMPLE LINE(0,0)–(119,31),PSET

REMARKS This statement is used to draw a straight line between any two points on the graphic screen specified by the SCREEN statement. When using the optional display controller, you can specify colour(s) also by setting a graphic screen on the external display with a SCREEN statement. In this case, you must pay attention to the range within which you can specify coordinates, as the screen configuration of the LCD is different from that of the external display.

PRESET is used to erase a line between two points. With the LCD screen, the line is simply erased. However, with the graphic screen set on the external display, the line is drawn in the background colour specified by the COLOR statement.

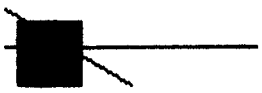
If coordinates 1 are omitted, coordinates 2 of the previous LINE statement or the coordinates specified by PSET or PRESET are assumed.

NOTE: LINE Statements may not work if the value specified for either of the two coordinates is more than 4000.

(See COLOR, PRESET and PSET.)

SAMPLE PROGRAMME

```
10 CLS
20 FOR I=5 TO 30
30 LINE (5,I)-(30,I),PSE
T
40 NEXT I
50 LINE (0,0)-(50,30),PS
ET
60 LINE (0,15)-(100,15),
PSET
70 COPY
80 END
```



LINE INPUT

FORMAT LINE INPUT["<prompt string>";]<string variable>
PURPOSE To input an entire line to a string variable.
EXAMPLE LINE INPUT "WHAT?";A$

REMARKS LINE INPUT inputs an entire line of up to 255 characters without the use of delimiters from the keyboard and assigns it to <string variable>. <prompt string> is a string literal that is displayed on the LCD screen before input. A question mark is not displayed unless it is part of the prompt string. All key inputs from the end of the prompt string to **RETURN** key are assigned to <string variable>. Therefore, you can input delimiters such as commas, double quotation marks, etc., which are not usually permitted by an INPUT statement.
A LINE INPUT may be escaped by pressing **BREAK** key and BASIC will return to command level. In this case, input a CONT command to resume programme execution.

(See INPUT.)

SAMPLE PROGRAMME

```
10 INPUT "NO. OF CUSTOME
RS";N
20 FOR I=1 TO N
30 LINE INPUT "CUSTOMER
DATA?";C$(I)
40 PRINT C$(I)
50 NEXT I
60 END

RUN
NO. OF CUSTOMERS? 2
CUSTOMER DATA? TOM
JONES, FIELDING AVE.
 TOM  JONES, FIELDING AV
E.
CUSTOMER DATA? DAVID
COPPERFIELD, DICKENS
ST.
 DAVID COPPERFIELD, DICK
ENS ST
```

LINE INPUT#

FORMAT LINE INPUT#<file number>,<string variable>
PURPOSE To read an entire line from a sequential data file to a 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 programme saved in ASCII mode is being read as data by another programme.

(See INPUT#.)

LIST/LLIST

FORMAT (1) LIST[<starting line number>][–[<ending line number>]]
(2) LLIST[<starting line number>][–[<ending line number>]]
PURPOSE To output a programme list (1) on the LCD or external display or (2) on the microprinter.
EXAMPLE
```
LIST 100–200
LIST –200
LIST 100–
LIST 200
LIST.
LIST
```

REMARKS LIST outputs all or part of the programme currently LOGged IN as specified by <starting line number> and <ending line number>. If only <starting line number> is specified, only that programme line will be output. If <starting line number> followed by a hyphen is specified, that line and all higher-numbered lines are listed. If <ending line number> preceded by a hyphen is specified, all lines from the beginning of the programme through that line are listed. If both line numbers are omitted, the entire programme is listed. After the occurrence of an error, or after programme editing, the line number can be replaced with a period. Listing can be interrupted by pressing **BREAK** key, or temporarily halted with **PAUSE** key. A LIST command may be written in a programme for listing, but the command following the LIST command will not be executed and the system will return to command level. The usage of LLIST is the same as that of LIST except that the list output is on the microprinter.

(See LOGIN.)

LIST <file descriptor>

FORMAT **LIST <file descriptor>[,[<line number>][–[<line number>]]]**

PURPOSE To output a programme list into a specified file.

EXAMPLE **LIST "COM0:**

REMARKS When a cassette is specified as the device name, this command functions the same as SAVE in ASCII format. If <file descriptor> is specified with a string expression or string variable, it always begins with double quotation marks (").
The line numbers are specified in the same manner as LIST except that line numbers must always be preceded by a comma.

(See LIST.)

*LIST "COM0:"

FORMAT **LIST"COM0:[(<BLPSC>)]"[, [<line number>][–[<line number>]]]**

PURPOSE To specify the interface conditions of the RS-232C port and execute LIST.

EXAMPLE **LIST"COM0:(2701B)"**

REMARKS The usage of this command is the same as that for LIST except that the output device is the RS-232C port.
For details of <BLPSC> which specifies the interface conditions, refer to OPEN"COM0:".

LOAD

FORMAT LOAD[<file descriptor>[,R]]
PURPOSE To load a programme file into the memory.
EXAMPLE LOAD"CAS1:PROG1.ASC"

REMARKS This command loads the programme 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 programme is run immediately after it is loaded.
If <file descriptor> is omitted, the first file of the default device is loaded.
LOAD retains the programmes 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 programme") error.

(See LOGIN, SAVE, STAT and TITLE.)

*LOAD COM0:"

FORMAT LOAD"COM0:[(<BLPSC>)]"
PURPOSE To specify the interface conditions of the RS-232C port and execute LOAD.
EXAMPLE LOAD"COM0:(68N2B)"[,R]

REMARKS The usage of this command differs from the normal LOAD command only in that a programme is read through the RS-232C port. However, the transmitted programme must be in ASCII format or an error will occur and the programme will not be read. For <BLPSC> which specifies the interface conditions, refer to OPEN"COM0:"

LOADM

FORMAT **LOADM<file descriptor>][,[offset value][,R]]**
PURPOSE To load a machine language programme file into the memory.
EXAMPLE **LOADM"CAS1:ABC"**

REMARKS The file to be LOADed should be a machine language programme 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, programme 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 programme 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 programme file, the CPU will interpret this file as a machine language programme file and will execute loading accordingly. If this happens, the BASIC programmes 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, even if 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 and 5.3, Machine Language Programmes.)

*LOAD?

FORMAT LOAD?[<file descriptor>]
PURPOSE To check files.
EXAMPLE LOAD?"CAS1:PROG1.ASC"

REMARKS LOAD? checks whether or not the data file specified by <file descriptor> has been correctly recorded. If <file descriptor> is omitted, "CAS0:" is assumed when the microcassette is connected to the HX-20, and "CAS1:", when it is not connected.
LOAD? command performs CRC check while reading the file (programmes and data) output to the auxiliary memory to confirm that the data file has been correctly recorded. If an error is found in the CRC check, an IO ("Device I/O") error will be displayed. As the data file is not actually loaded during the CRC check, there is no danger that the programs stored in the memory will be lost. This command is not intended to verify the contents of the cassette tape with the contents of the memory. For this reason, LOAD? can be executed even if there is no target programme in the area currently logged in.

When executing LOAD? against files (CAS1: CAS0:) stored on cassette tape, the tape will be searched until the file specified by <file descriptor> is found and then actual check begins. After the execution of this command, the position of the tape recorder head will be at the end of the specified file; namely, at the beginning of the next file.

*LOCATE

FORMAT **LOCATE<horizontal coordinate>,<vertical coordinate>**
[,<cursor switch>]

PURPOSE To specify the cursor position on the screen.

EXAMPLE **LOCATE 10, 10, 0**

REMARKS LOCATE positions the cursor at a specified position on the virtucal screen. Both <horizontal coordinate> and <vertical coordinate> must be specified. The cursor cannot be positioned outside the bounds of the virtual screen specified by a WIDTH command.

If the cursor is positioned outside the physical screen by executing a LOCATE command, the physical screen will automatically move to follow the cursor, so that the cursor is positioned at the upper left-hand corner of the screen. However, if the physical screen is at the right-hand end or the trailing end of the virtual screen, the cursor may not be at the upper left-hand corner. Even if the physical screen has been fixed at the left-hand end of the virtual screen (mode 0), the execution of a LOCATE command will cause the physical screen to move from that position (mode 1). (When the system returns to command level after the execution of a programme, the physical screen is put in mode 0.)

The cursor is turned off if the option <cursor switch> is specified as "0" and is turned on if specified as "1".

(See SCROLL.)

*LOCATES

FORMAT LOCATES <horizontal coordinate>, <vertical coordinate>
PURPOSE To specify the position of the physical screen.
EXAMPLE LOCATES 0, 0

REMARKS LOCATES moves the physical screen so that its upper left-hand corner will be in the location on the virtual screen specified by <horizontal coordinate> and <vertical coordinate>. The physical screen cannot leave the bounds of the virtual screen.
As with a LOCATE command, after the execution of a LOCATES command, the screen is always put in mode 1. Even after the execution of a LOCATES command, the cursor position on the physical screen will remain unchanged.

(See LOCATE)

*LOGIN

FORMAT LOGIN <expression>[,R]
PURPOSE To switch the programme areas.
EXAMPLE LOGIN 3

REMARKS In BASIC, the memory space is divided into five areas, each capable of storing separate programmes. A LOGIN command specifies the programme areas to be used (for programme execution, programme editing, etc.)
<expression> must be an integer between 1 and 5.
All commands for programme execution and modification (NEW, LIST, LOAD, SAVE, etc.) are effective only in the areas which have been specified by LOGIN commands. If the <R> option is specified, the programme execution begins soon after the area for the programme was switched. After the execution of a LOGIN command, all variables are cleared.

(See STAT.)

*MEMSET

FORMAT MEMSET [<bottom address of memory>]
PURPOSE To specify the lower limit of the memory.
EXAMPLE MEMSET &HOD00

REMARKS In BASIC, programmes written in machine language are placed before the BASIC programme text area. To enable the storage of machine language and BASIC programmes at the same time, the lower limit of the memory to be used by BASIC must be set using a MEMSET command. This also sets the memory locations for machine language programmes.
At cold start, <bottom address of memory> is set at &H0A3F. Warm start will not affect this <bottom address of memory>.
If <bottom address of memory> is omitted, &H0A3F is assumed. Upon execution of a MEMSET command, all variables are cleared.
Addresses 0 through &H0A3F are allocated as the I/O and system areas and are not permitted for the storage of machine language programmes.

SAMPLE PROGRAMME

```
100 ON ERROR GOTO 200
110 FOR I=&H0A3F TO &H0B
00
120 POKE I,0
130 NEXT I
140 END
200 PRINT "BASIC PROGRAM
ME AREA. TO PROTECT BASI
C, POKE CANNOT BE EXECUT
ED."
210 RESUME 140
MEMSET &H0AFF
RUN

BASIC PROGRAMME AREA. TO
 PROTECT BASIC, POKE CAN
NOT BE EXECUTED.
```

MERGE

FORMAT MERGE [<file descriptor>[,R]]

PURPOSE To merge a specified programme file into the programme currently in memory.

EXAMPLE "CAS1:PROG3.ASC"

REMARKS MERGE command merges the programme file specified by <file descriptor> into the programme 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 programme in the memory, the lines in the file will replace the corresponding lines in the memory. If the option R is specified, the merged programme 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.)

*MERGE "COM0:"

FORMAT MERGE "COM0:[(<BLPSC>)"[,R]]

PURPOSE To specify the interface conditions of the RS-232C port and to execute MERGE.

EXAMPLE MERGE"COM0":(68N2B)",R

REMARKS This command is essentially the same as the normal MERGE command except that the programme is read via the RS-232C port.

For details of <BLPSC> which specifies the interface conditions of the RS-232C port, see OPEN "COM0:"

MID$

FORMAT **MID$ (<string exp 1>,<n>[,<m>])=<string exp 2>**
where n and m are integer expressions and <string exp 1> and <string exp 2> are string expressions.

PURPOSE To replace a portion of one string with another string.

EXAMPLE **MID$(A$,2)="BASIC"**

REMARKS MID$ replaces the characters in <string exp 1>, beginning at position <n>, by the characters in <string exp 2>. The optional <m> efers to the number of characters from <string exp 2> that will be used in the replacement. If <m> is omitted, all of <string exp 2> is used. However, regardless of whether <m> is omitted or included, the replacement of characters never goes beyond the original length of <string exp 1>.
As the length of <string exp 1> never changes, the value of <n> cannot exceed the number of characters in (string exp 1), nor can <n> be a negative value. <string exp 1> cannot be a null string.

(See MID$ function)

SAMPLE PROGRAMME

```
100 FOR I=1 TO 20 STEP 4
110 A$="----------------
----"
120 MID$(A$,I)="*"
130 PRINT A$:NEXT
140 END
```

```
*---------------
----*-----------
--------*-------
------------*---
----------------*
```

MON

FORMAT MON
PURPOSE To transfer programme control to the machine language monitor.
EXAMPLE MON

REMARKS MON is used to transfer programme control from BASIC to the built-in machine language monitor. The machine language monitor commands are as follows:

S<address> Changes the contents of the memory. Input of a period (".") terminates this command.

D<address> Displays the contents of the memory.

G<execution address>,<breakpoint> Causes a programme to be executed up to <breakpoint>. The HX-20 then returns to Monitor.

K<string>^:@ Writes a menu number at the beginning of <string>, then writes a list of strings up to ^:@. When the power is switched on, the program specified by the menu number will be automatically selected and the same operations will be performed as if the character strings were input from the keyboard. <string> can be a maximum of 17 characters.
This function is cancelled upon input of K^:@, (which defines a null string for <string>).

B Causes HX-20 to escape from Monitor mode.

X Changes the contents of the registers. Each time **RETURN** key is pressed, the content of each register is displayed. By pressing **RETURN** key following the input of a numeric value, the contents of the register can be changed. This command is terminated upon input of a period.

R<device>,<filename>, R Loads a specified file into the memory from a specified device. If R is used with this command, the programme is run after loading the specified file.

V<device>,<filename> Performs CRC check of a specified file in a specified device.

W<device>,<filename> Saves a specified file in a specified device.

A Specifies addresses when R, V or W command is used.

Each time **RETURN** key is pressed, 2-byte data is requested in the following order.

T Top address of memory
L Bottom address of memory
O Offset
E Execution starting address

For <device>, specify C (Cassette), M (Micro-cassette), or P (ROM cartridge). However, with W command, P (ROM cartridge) cannot be specified. <filename> consists of a filename of eight or less characters and a three-character filetype.
<filename> = <filename>. <filetype>
For details, refer to Chapter 10, How to Use the Monitor in the HX-20 Operation Manual.

MOTOR

FORMAT **MOTOR [<switch>]**
PURPOSE To turn ON/OFF the motor of the external audio cassette.
EXAMPLE **MOTOR ON**

REMARKS MOTOR turns on or off the Remote terminal of the external audio cassette connected to the HX-20. By specifying <switch> as either ON or OFF, the motor of the external audio cassette can be controlled. If <switch> is omitted, the motor will reverse its ON/OFF state. In other words, if the Remote terminal is in the OFF state, the motor will reverse from OFF to ON and vice versa.

*NEW

FORMAT NEW
PURPOSE To delete the programme in the memory and clear all variables.
EXAMPLE NEW

REMARKS NEW deletes all programmes in the programme area currently LOGged IN. When this command appears in a programme, the programme in that area is cleared and BASIC returns to command level. If the programme is in the area that has been named by a TITLE statement, execution of a NEW command will result in a PP ("Protected programme") error. Execution of a NEW command causes all files currently open to be closed.

(See LOAD, LOGIN, and TITLE.)

ON ERROR GOTO

FORMAT ON ERROR GOTO <line number>
PURPOSE To enable error trapping and specify the first line of the error handling subroutine.
EXAMPLE ON ERROR GOTO 1000

REMARKS An ON ERROR GOTO statement transfers programme control to a specified error handling subroutine if an error occurs during the execution of a program. If an error is detected with this statement being executed, BASIC will execute a programme beginning with the line specified by <line number> and will not display an error message. Therefore, by enabling error trapping with that programme, you can prevent programme execution from being halted due to the occurrence of an error.
To disable error trapping, execute an ON ERROR GOTO 0. If an error is encountered for which there is no recovery action, execute an ON ERROR GOTO 0 in an error handling subroutine and an error message will be displayed and programme execution will be terminated.

(See RESUME, ERL/ERR, and APPENDIX A, Error Messages.)

SAMPLE PROGRAMME

```
100 ON ERROR GOTO 200
110 INPUT "A=",A
120 IF A<0 THEN ERROR 25
0
130 IF FIX(A)<>A THEN ER
ROR 255
140 B#=1#
150 FOR I=2 TO A
160 B#=B#*I:NEXT I
170 PRINT A;"!"
180 PRINT "IS ";B#
190 GOTO 100
200 '***ERROR***
210 IF ERR=255 AND ERL=1
30 GOTO 250
220 IF ERR=6 AND ERL=160
 GOTO 270
230 IF ERR=250 AND ERL=1
20 GOTO 290
240 ON ERROR GOTO 0
250 PRINT "INPUT AN INTE
GER "
260 RESUME 110
270 PRINT "VALUE TOO LAR
GE "
280 RESUME 110
290 PRINT "NEG NUMBER NO
T ACCEPTED "
300 RESUME 110
310 END
```

```
RUN
A=10
 10 !
IS  3628800
A=-30
NEG NUMBER NOT ACCEPTED
A=17
 17 !
IS  355687428096000
A=45
VALUE TOO LARGE
```

*OPEN"COM0:"

FORMAT **OPEN "<mode>", [#]<file number>, "COM0:[(<BLPSC>)]"**
PURPOSE To specify the interface conditions for the RS-232C port and execute OPEN.
EXAMPLE **OPEN "0", #1,"COM0:(68N2B)"**

REMARKS This command is essentially the same as the normal OPEN command except that it specifies the interface conditions of the RS-232C port. <BLSPC> consists of 5 characters each specifying one of the interface conditions of the RS-232C port as follows.

B (Bit rate)
Numerics 0 to 6 are used to specify the bit rate (data transfer rate).

0: 110 bps
1: 150 bps
2: 300 bps
3: 600 bps
4: 1,200 bps
5: 2,400 bps
6: 4,800 bps

L (Word length)
Either 7 or 8 is used to specify the word length of 1-character data.

7: 7 bits/character
8: 8 bits/character

P (parity)
N, E, or O is used to specify the method of parity check.

N: No parity check
E: Even parity check
O: Odd parity check

S (Stop bits)
Either 1 or 2 is used to specify the stop bit length.

1: 1-bit length
2: 2-bit length

C (Control line active)
Active control (signal) lines are determined using hexadecimal digits 0 through F (corresponding to 4 binary bits, with each bit corresponding to one control line). In the following chart, active signal lines are represented by "○" and inactive lines (to be ignored) by "×". For the RTS signal, "+" sign indicates that the positive signal potential is active and "−" sign indicates that the negative signal potential is active.

Control line / Hexadecimal number	CTS	DSR	RTS	CD
0	○	○	−	○
1	○	○	−	×
2	○	○	+	○
3	○	○	+	×
4	○	×	−	○
5	○	×	−	×
6	○	×	+	○
7	○	×	+	×
8	×	○	−	○
9	×	○	−	×
A	×	○	+	○
B	×	○	+	×
C	×	×	−	○
D	×	×	−	×
E	×	×	+	○
F	×	×	+	×

<BLPSC> can be omitted. If omitted, the HX-20 defaults to the values set last time. At warm start, HX-20 is set in the following conditions.

Bit rate: 4,800 bps
Word length: 8 bits/character
Parity: No parity check
Stop bit length: 2 bits
CTS: Ignore
DSR: Active
RTS: + potential is active.
CD: Ignore

Using the RS-232C port, two files (for input and output) can be opened at the same time. In this case, different interface conditions cannot be specified for the files to be opened. The conditions under which the first file was opened remain in effect.

OPTION BASE

FORMAT OPTION BASE [0 | 1]

PURPOSE To declare the minimum value for array variable subscripts.

EXAMPLE OPTION BASE 1

REMARKS OPTION BASE is used to declare the minimum value for array variable subscripts as either 0 or 1. The default base is 0. However, if an OPTION BASE 1 is executed, the minimum value for a subscript is set to 1. Thereafter, if an array element is referenced with a subscript specified as 0, a BS ("Bad subscript") error will occur. This declaration cannot be made after array variables have been declared in a programme or referenced. Also, once declared, this minimum value cannot be altered by redeclaration, which will cause a DD ("Duplicate definition") error to occur.
Once an OPTION BASE statement has been executed, the declaration by the OPTION BASE can be neither changed nor cancelled until a RUN or CLEAR command is executed.

(See CLEAR and DIM.)

SAMPLE PROGRAMME

```
100 ON ERROR GOTO 160
110 OPTION BASE 1 : DIM
A(9)
120 FOR I=0 TO 9
130 A(I)=I:PRINT A(I);
140 NEXT I
150 OPTION BASE 0:END
160 IF ERR=9 THEN PRINT
"MINIMUM SUBSCRIPT IS 1"
 : RESUME 140
170 IF ERR=10 THEN PRINT
 "OPTION BASE CANNOT BE
CHANGED"
180 RESUME NEXT
RUN
MINIMUM SUBSCRIPT IS 1
 1  2  3  4  5  6  7  8
 9
OPTION BASE CANNOT BE CH
ANGED
```

*PCOPY

FORMAT **PCOPY <expression>**
PURPOSE To copy BASIC programme into another programme area.
EXAMPLE **PCOPY 3**

REMARKS PCOPY command copies the programmes in the currently logged-in programme area into another programme area specified by <expression>. <expression> must be within the range of 1 to 5. If there is already a programme in the programme area specified by <expression> or if no programme exists in the currently logged-in programme area, execution of a PCOPY will cause an FC ("Illegal function call") error. If the programme to be copied is too large, an OM ("Out of memory") error occurs and PCOPY is not executed.

(See LOGIN, STAT, and TITLE.)

*POKE

FORMAT POKE <address>, <numeric expression>
PURPOSE To write a byte into a specified memory location.
EXAMPLE POKE &HOCOO, &H39

REMARKS POKE command writes one byte (8 bits) of data specified as <numeric expression> into a memory location specified by <address>.
<numeric expression> must be a one-byte integer expression in the range of 0 to 255 (&HO to &HFF). POKE rounds off less significant digits to the nearest integer (e.g., 1.5 to 2 and 1.4 to 1).
<address> must be a two-byte integer expression in the range of 0 to 65535 (&HO and &HFFFF). POKE rewrites the data currently in the memory.
Note that careless use of this command can be a source of malfunctions. Since &H0 to &H4D comprise a special area allocated for input/output, an overrun may occur just by reading it.
BASIC programmes are stored in the area above the address specified by a MEMSET command. For this reason, any attempt to execute a POKE statement against the abovementioned special area will cause an FC error to occur. To rewrite any of these addresses, you must write &H80 into address &H7E. In the area between addresses &H4E and &HA3F, execution of a POKE statement will not cause any error.
However, as this area is used as a work area by BASIC, rewriting any of the values in these memory locations may result in malfunctioning of BASIC. Therefore, please restrict execution of POKE statements to the machine language area between address &HA40 and the address specified as the bottom address of memory by a MEMSET statement. The complementary function to POKE is PEEK, a function which reads <numeric expression> from a specified memory location.
POKE and PEEK are useful for efficient data storage, loading machine language subroutines, and passing arguments and results to and from machine language subroutines.

(See PEEK, USR, and 5.3 Machine Language Subroutines.)

Character set selection

Character set selection can be made using the POKE and EXEC statements.
To select the character set, POKE the address &H7F and write the corresponding data shown below:

Character set of country to be selected	Data to be written
Spain	&H10
Italy	&H11
Sweden	&H12
Denmark	&H13
England	&H14
Germany	&H15
France	&H16
U.S.A.	&H17

The starting address for the EXEC statement is &HFF6A.

In order to return to the default character set (i.e., character set selected by the DIP switch), execute the following statements.

```
10 POKE &H7F,0
20 EXEC &HFF6A
```

Example: Spain

```
10 POKE &H7F,&H10
20 EXEC &HFF6A
```

PRESET

FORMAT **PRESET (<horizontal coordinate>, <vertical coordinate>)**
PURPOSE To erase a dot on a graphic screen.
EXAMPLE **PRESET (40,25)**

REMARKS PRESET resets the colour of any of the dots drawn on the graphic screen by a PSET or LINE statement to the background colour.
On the LCD, this command simply erases the specified dot from the graphic screen. If an external display is connected to the HX-20 using the optional display controller, this command will reset the specified dot to the background colour specified by a COLOR statement when the external display is set in the colour graphic mode by a SCREEN statement. As the LCD and the external display differ from each other in screen configuration, please pay attention to the range within which you can specify <coordinates>.

(See LINE, COLOR, PSET, and SCREEN.)

PRINT/LPRINT

FORMAT PRINT / LPRINT [<expression>[, / ;] <expression>...]]

PURPOSE To output data on the screen or the built-in microprinter.

EXAMPLE PRINT "EPSON"

REMARKS PRINT causes the data specified by <expression> to be output on the screen, while LPRINT command causes the same data to be output on the built-in microprinter. <expression> may be a numeric or string expression. If string expressions are used, character strings resulting from the expressions are output. If numeric expressions are used, the values resulting from the expressions are output. In this case, however, a blank space is left before and after each output value.

If the value is negative, a negative sign "−" is output in the blank preceding the value. When writing a list of expressions, the expressions must be separated by a comma, a semicolon or a blank. Delimiters can be omitted between a variable and a string constant and between one string constant and another. In this case, the effect is the same as when each expression was separated by a semicolon. A question mark ("?") may be used in place of PRINT in a PRINT statement, while LPRINT can neither be replaced nor omitted.

Output Format and Punctuation

- When the items in a list of expressions are separated with blanks or semicolons, the next value will be displayed or printed immediately after the last value. When the list of expressions are separated with commas, BASIC divides the line into display or print zones of 14 spaces each and outputs the value of each expression at each zone.
  If the result of an expression overlaps two or more zones, the next value will be output at the beginning of the zone following the last value.
- If the list of expressions terminates without a comma or semicolon, a carriage return is effected at the end of the line. If a comma or semicolon terminates the list of expressions, the next PRINT statement begins output on the same line, spacing accordingly.
- If the string to be output is longer than one line specified for the terminal, output continues on the next line. If all the strings or values to be output cannot be accommodated on a single line (between the cursor position and the end of the line), a carriage return is effected and output continues on the next line.

(See PRINT USING/LPRINT USING, SPC, and TAB.)

PRINT USING/LPRINT USING

FORMAT |PRINT |USING <"format string">;[<expression>[|;|
|LPRINT| |,|
<expression> ...]]

PURPOSE To output strings or numerics using a specified format.

EXAMPLE PRINT USING "####"; A,B

REMARKS PRINT USING/LPRINT USING determines the field and format of <expression> to be output by <"format string">. PRINT USING is used to display strings or numerics on the screen, while LPRINT USING is used to output the same on the microprinter.

String Fields

! Specifies that only the first character in the given string is to be output.

\ (n Blanks) \ . . . Specifies that (n+2) characters from the beginning of the given string will be output. If the string is longer than the field, the extra characters are ignored.
If the field is longer than the string, the string will be left-justified in the field and padded with spaces on the right. (See Note below.)

& Used to output character strings. When a number of & is specified, the value of a variable assigned to each & will be output. If the number of "&" is greater than the number of strings in the expression list, the extra "&" will be ignored.

Numeric Fields

A number sign is used to represent each digit position. Digit positions are always filled. If the number to be output has fewer digits than positions specified, the number will be right-justified (preceded by spaces) in the field. (See Note below.)

. A decimal point may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, the digit will always be output as 0. Numbers are rounded as necessary. In the last example, three spaces were inserted at the end of the format string to separate the printed values on the line.

+ A plus sign at the beginning or end of a format string will cause the sign of the number (plus or minus) to be output before or after the number. If two or more plus signs are placed in succession, extra plus signs will be handled as "Characters Other Than Formatting Characters" described later.

- A minus sign at the end of a format string will cause negative numbers to be output with a trailing minus sign. If a minus sign is placed at the beginning of a format string, or if two or more minus signs are placed in succession, the minus signs will be handled as "Character Other Than Formatting Characters" described later.

** A double asterisk at the beginning of a format string causes leading spaces in the numeric field to be filled with asterisks. The ** also specifies positions for two more digits.

$$ A double dollar sign at the beginning of a format string causes a dollar sign to be output to the immediate left of the formatted number. The $$ specifies two more digit positions, one of which is the dollar sign. The exponential format ("^^^^") described below cannot be used with $$. (See Note below.)

$ The **$ at the beginning of a format string combines the effects of the above two symbols ("" and "$$"). Leading spaces will be asterisk-filled and a dollar sign will be output before the number. **$ specifies three more digit positions, one of which is used as the output area for a dollar sign.

, A comma that is to the left of the decimal point in a format string causes a comma to be output to the left of every third digit to the left of the decimal point. If a comma is placed to the right of the decimal point in a format string, a comma is output at the end of the formatted number.

^^^^ Four carets (or up-arrows) may be placed after the digit position ("#") characters to specify an exponential format. (See Note below.)

_ Used to represent any of the abovementioned formatting characters as a literal character. An underscore in a format string causes the next character to be output as a character which has no formatting function.

% If the number to be output is larger than the specified numeric field, a percent sign is output in front of the number. If rounding causes the number to exceed the field, a percent sign will be printed in front of the rounded number.

Characters Other Than Formatting Characters

If any characters other than the abovementioned formatting characters (e.g., alphanumerics, graphic symbols, etc.) are placed at the beginning or end of a format string, such characters will be output before or after the formatted number.

NOTE: The formatting characters shown above apply to the ASCII character set. If your selected character set is other than ASCII, some of the formatting characters will be output differently as shown below.

USASCII	France	Germany	England	Denmark	Sweden	Italy	Spain
#	#	#	£	#	#	#	Pt
$	$	$	$	$	¤	$	$
\	ç	Ö	\	ø	Ö	\	Ñ
^	^	^	^	^	Ü	^	^

(See Section 2.4, Character Sets, for detailed information.)

SAMPLE PROGRAMME

```
100 A$="EPSON HX-20"
110 B=123.456
120 C=-123.456
130 D=1234.56
140 E=123456
150 PRINT USING "!";A$
160 PRINT USING "\      \
";A$
170 PRINT USING "THIS IS
 A &";A$
180 PRINT USING "###";B
190 PRINT USING "###.#";
B
200 PRINT USING "+##.#";
B
210 PRINT USING "###.#-"
;C
220 PRINT USING "**#.#";
B
230 PRINT USING "$$#.#";
B
240 PRINT USING "**$.#";
B
250 PRINT USING "###,.#^
^^^^";D
260 PRINT USING "$$#._-"
;D
270 PRINT USING "###";E
280 END
RUN
E
EPSON H
THIS IS A EPSON HX-2
0
123
123.5
%+123.5
123.5-
123.5
%$123.5
%$123.5
 123.5E+01
 123.5E+01
%$1235.-
%123456
>
```

*PRINT#

FORMAT PRINT# <file number>, [<expression>...]
PURPOSE To write data into a sequential file.
EXAMPLE PRINT#1,A,B

REMARKS <file number> is the number used when the file was OPENed for output. <expression> is the numeric or string expression which is to be written into the file. For CAS0: and CAS1:, PRINT#, unlike output to the display screen, automatically delimits the data before writing it into the file. Therefore, irrespective of whether commas or semicolons are used as delimiters in the list of expressions, data will be output in the same format.

(See INPUT#, PRINT/LPRINT USING, and PRINT# USING.)

PRINT# USING

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

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

(See OPEN, PRINT USING, and PRINT#)

PSET

FORMAT PSET (<horizontal coordinate>, <vertical coordinate>)[,<colour>]

PURPOSE To draw dots on a specified graphic screen.

EXAMPLE PSET (30,20)

REMARKS PSET draws a dot at a specified location on the graphic screen specified by a SCREEN statement. PSET does not affect the dots already drawn on the screen. If the optional display controller is used, please pay attention when you specify coordinates, as the external display is different from the LCD in screen configuration. If you set the external display in colour graphic mode using a SCREEN statement, <colour> can also be specified. In this case, if the colour of the existing dots is different from your selected <colour>, it will change to the newly specified colour.
If <colour> is omitted, the colour of the dot to be drawn on the screen will be the same as the foreground colour specified in a COLOR statement.

(See COLOR, PRESET, and SCREEN.)

SAMPLE PROGRAMME

```
LIST

100 CLS:PI=3.14159
110 DEFINT X,Y
120 FOR I=0 TO 2*PI STEP PI/54
130 X=COS(I)*15+64
140 Y=SIN(I)*15+16
150 PSET(X,Y)
160 NEXT:COPY
>
```



*PUT%

FORMAT PUT% <record number>, <variable>[,<variable>...]
PURPOSE To write the values of variables into a RAM file.
EXAMPLE PUT%0, A!, B#, C$

REMARKS PUT% writes the values of variables into the record specified by <record number>. String variables must be written at the end of the variable list and only one string variable per record is permitted.

If all the values of the string variables cannot be written into the specified record, the excess portion of the string variables will be ignored.
If there is an extra capacity within the record after the last variable has been written, the record will be padded with spaces.

(See DEFFIL, GET% and RAM Files.)

SAMPLE PROGRAMME

```
100 DEFFIL 20,0
110 A=1.2:B%=3:C$="ABCDE
FG"
120 PUT%0,A,B%,C$
130 D=0.123:E$=C$+C$
140 PUT%1,A,B%,D,E$
150 'GET% CAN BE EXECUTE
D EVEN FOR TYPES DIFFERE
NT THAN THE PUT% STATEME
N BUT THE VALUE WILL BE
REFUSED
160 FOR I=0 TO 1
170 PRINT
180 GET% I,J,K%,L$
190 PRINT I;J;K%;L$
200 GET% I,J,K%,M,M$
210 PRINT I;J;K%;M;M$
220 NEXT I
230 END

 0  1.2  3
ABCDEFG
 0  1.2  3  8.22735E-20
EFG

 1  1.2  3
>< mABCDEFGABC
 1  1.2  3  .123
ABCDEFGABC
```

RANDOMIZE

FORMAT RANDOMIZE [<expression>]
PURPOSE To reseed the random number generator.
EXAMPLE RANDOMIZE

REMARKS RANDOMIZE changes the sequence of random numbers by supplying the random number generator with a new seed. The random number seed must be specified by <expression> within the range −32768 to 32767. If <expression> is omitted, a "Seed?" message will be displayed upon execution of RANDOMIZE, asking for the value of a random number seed. If the value input is outside the above random number seed range, an error will occur. RANDOMIZE statements cannot be executed in direct mode.

READ

FORMAT READ<variable>[,<variable>...]
PURPOSE To read values from a DATA statement and assigning them to variables.
EXAMPLE READ A, I, C$

REMARKS A READ statement must always be used in conjunction with a DATA statement. READ statements assign variables to DATA statement values on a one-to-one basis. READ statement variables may be numeric or string, and the values read must agree with the variable types specified. If they do not agree, a "Syntax error" will result.
If the number of variables in the list of variables in the READ statement(s) exceeds the number of elements in the DATA statement(s), an OD ("Out of data") error will occur.
If the number of variables specified is fewer than the number of elements in the DATA statement(s), subsequent READ statements will begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

(See DATA and RESTORE.)

REM

FORMAT REM[<remark>]
PURPOSE To allow explanatory remarks to be inserted in a program.
EXAMPLE REM COMMENT MESSAGE

REMARKS A REM statement is a nonexecutable statement that is output exactly as it was entered in the programme.
In a REM statement, keyword "REM" can be replaced with an apostrophe ("'").
In a REM statement, a colon (":") is recognized as part of a <remark>. Therefore, no statement can be placed after the REM statement.

RENUM

FORMAT RENUM [<new line number>][,[<old line number>][,[<increment>]]
PURPOSE To renumber programme lines.
EXAMPLE RENUM

REMARKS <new line number> is the first line number to be used in the new sequence. The default is 10. <old line number> is the line in the current programme where renumbering is to begin. The default is the first line of the program. <increment> is the increment to be used in the new sequence. The default is 10.
RENUM also changes all line number references following GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB and ERL statements to reflect the new line numbers. If a line number nonexistent in the programme appears after one of these statements, the error message "Undefined Line Number nnnn in mmmm" is output. The incorrect line number reference (nnnn) is not changed by RENUM, but line number mmmm may be changed.
A RENUM command cannot be used to change the order of programme lines or to create line numbers greater than 64000. An FC ("Illegal function call") error will result if RENUM is used contrary to the above rules. In this case, the programme line numbers will remain unchanged. RENUM is valid only in the programme area currently LOGged IN.

RESTORE

FORMAT RESTORE [<line number>]
PURPOSE To allow DATA statements to be reread from a specified point.
EXAMPLE RESTORE 1000

REMARKS RESTORE causes the next READ statement to access the first item in the DATA statement at the line specified by <line number>. If <line number> is omitted, the next READ statement will read from the first DATA statement in the programme.
If there is no DATA statement at the line specified by <line number>, the READ statement will read from the DATA statement at the lowest line number after the specified <line number>.
If the line specified by <line number> does not exist in the programme, a UL ("Undefined line number") error occurs.

(See DATA and READ.)

SAMPLE PROGRAMME

```
LIST

100 DATA "**************","   HC -",2  0
110 READ A$
120 PRINT A$
130 READ A$,B%
140 PRINT A$;B%
150 RESTORE 100
160 READ A$
170 PRINT A$
180 END
RUN
**************
   HC - 20
**************
>
```

RESUME

FORMAT RESUME [| NEXT |]
| <line number> |

PURPOSE To continue programme execution after an error recovery procedure has been performed.

EXAMPLE **RESUME 100**

REMARKS With RESUME, you can specify the statement or programme line at which programme execution is to resume after an error recovery procedure has been completed. The three format options are as follows.

RESUME	Execution resumes at the statement which caused the error.
RESUME NEXT	Execution resumes at the statement immediately following the one which caused the error.
RESUME<line number>	Execution resumes at the line specified by <line number>.

(See ON ERROR GOTO.)

RUN

FORMAT (1) **RUN [<line number>]** or
(2) **RUN <file descriptor>[,R]**
PURPOSE To start programme execution.
EXAMPLE (1) **RUN 300**
(2) **RUN"CAS0:PROG4.ASC"**

REMARKS In format (1), if <line number> is specified, execution of the programme currently in the memory begins on that line. If <line number> is omitted, execution starts at the lowest line number.
In format (2), a programme specified by <file descriptor> is loaded into the memory and then programme execution begins.
Execution of a RUN command clears all variables and closes all open files before loading the designated programme. However, if the <R> option is used with this command, all data files remain OPEN.
Before executing a RUN command in format (2), use a STAT command to check the programme area currently LOGged IN.

(See LOGIN and STAT.)

RUN"COM0:"

FORMAT **RUN"COM0:[(<BLPSC>)]"[,R]**
PURPOSE To specify the interface condition of the RS-232C port and execute RUN.
EXAMPLE **RUN"COM0:(68N2B)"**

REMARKS Except for the fact that the programme is loaded into memory via the RS-232C port, this command is essentially the same as the normal RUN command. However, the programme to be transmitted through the port must be in ASCII format.
For details of <BLPSC>, which specifies the interface conditions of the RS-232C port, refer to OPEN"COM0:".

SAVE

FORMAT SAVE <file description>[,|A|V|]

PURPOSE To save an EPSON BASIC programme on a specified file.

EXAMPLE SAVE"CAS0:ABC"

REMARKS This command is used to save BASIC programmes on the file specified by <file descriptor>. If a file already exists under the specified filename, the programme can be saved on another location under the same filename. If this is done, two different programmes saved cannot be distinguished from each other as they are under the same filename. Be careful not to use the same filename.
The <A> option saves a file in ASCII format. If <A> is not specified, the file is saved in a compressed binary format. ASCII format requires more space than the binary format but some file access requires that the file be in ASCII format (e.g., when using the MERGE command for programme editing). Also, a file saved in ASCII format may be read as a data file.

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 programme and program verification (CRC check) will be performed. If <device> other than the microcassette recorder is specified, the <V> option will be ignored.

(See LOAD and MERGE.)

*SAVE"COM0:"

FORMAT **SAVE"COM0:[(<BLPSC>)]",A**
PURPOSE To specify the interface conditions of the RS-232C port and execute SAVE.
EXAMPLE **SAVE"COM0: (68E13)",A**

REMARKS Except for the fact that the programme is output via the RS-232C port, this command is essentially the same as the normal SAVE command.
If A is omitted, no error will occur. However, since the data in a compressed binary format cannot be read properly by the RS-232C port, an attempt to transmit such data will be meaningless.
For details of <BLPSC>, which specifies the interface conditions of the RS-232C, refer to OPEN"COM0:".

*SAVEM

FORMAT **SAVEM <file descriptor>,<top address>,<bottom address>, <execution starting address>[,V]**
PURPOSE To save the memory contents on a specified file.
EXAMPLE **SAVEM"CAS1:ABC", &H0B00, &H0C00, &H0B00**

REMARKS SAVEM saves a machine language programme 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, programme 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 programme loaded into the memory by a LOADM command will begin at <execution starting address>. Even if the data saved is not a machine language programme, <execution starting address> cannot be omitted and the same value as the <top address> must be set.

(See LOADM.)

*SCREEN

FORMAT **SCREEN <text>, <graphic mode>**
PURPOSE To specify the text or graphic screen mode.
EXAMPLE **SCREEN 0,2**

REMARKS SCREEN sets the LCD or external display in the text screen mode and/or graphic screen mode. (The optional display controller is required to connect the HX-20 to an external display.)
The value of <text> must be either 0 or 1. Specify 0 to display a text screen (physical screen) on the LCD and 1 to display a text screen on the external display.

The value of <graphic mode> must be in the range of 0 to 2. Specify 0 to display a graphic screen on the LCD. If 1 is specified, the external display is set in colour graphic mode (128×64 dots, 4-colour). If 2 is specified, the external display is set in high-resolution mode (128×96 dots, black and white). At the time of warm start, the LCD is set in both text and graphic screen mode.

SCREEN 0,0 ... Displays both text and graphic screens on the LCD.
SCREEN 0,1 ... Displays a text screen on the LCD and a graphic screen in the colour mode on the external display.
SCREEN 0,2 ... Displays a text screen on the LCD and a graphic screen in high-resolution mode on the external display.
SCREEN 1,0 ... Displays a text screen on the external display and a graphic screen on the LCD.

Due to the limited capability of the display controller, both text and graphic screens cannot be displayed simultaneously on the external display.

*SCROLL

FORMAT **SCROLL[<speed>][,[<mode>][,<scroll step X>,<scroll step Y>]]**

PURPOSE To specify the SCROLL function of the physical screen.

EXAMPLE **SCROLL 9,0,10,4**

REMARKS <speed> specifies the speed when the LCD screen scrolls using a value in the range of 0 to 9, with 9 indicating the highest scrolling speed and 0, the lowest. However, the scrolling speed of the external display is fixed and not adjustable.

<mode> specifies the manner of movement of the physical screen and its value must be either 0 or 1.
If you specify <mode> as 0, the physical screen is fixed at the left-hand end of the virtual screen and can move only up or down. In this mode, if the cursor position is outside the bounds of the physical screen, note that it will not come back into view unless the physical screen is moved, even if an output command is executed.
If you specify <mode> as 1, the physical screen will automatically move to follow the cursor.
When the system returns to command level after the execution of a programme, <mode> is set to 0.
<scroll step X> indicates the number of characters for which the screen should move at one time when CTRL + [ ] or CTRL + [ ] are typed. The value of <scroll step X> must be in the range of 1 to 20 for the LCD and 1 to 32 for the external display. <scroll step Y> indicates the number of lines the screen should move vertically at one time when CTRL + P, CTRL + Q or SC ; RN are typed. The value of <scroll step Y> must be in the range of 1 to 4 for the LCD and 1 to 16 for the external display. The default values at warm start are as follows:

For the LCD	SCROLL 9, 0, 10, 4
For the external display	SCROLL 9, 0, 16, 16

*SOUND

FORMAT SOUND <tone>,<duration>
PURPOSE To sound a specified tone.
EXAMPLE SOUND 10, 10

REMARKS SOUND causes the built-in piezoelectric speaker to sound at a specified <tone> for a specified <duration>.
<tone> may be a value in the range 0 to 56. 0 indicates a pause and the range of 1 to 28 corresponds to the scale from tone C (do) to tone B (ti) which is four octaves higher than the first C. 13 is equivalent to 880 Hz. The range of 29 to 56 corresponds to the scale with tones each of which is a half tone higher than those in the scale represented by 1 to 28.
<duration> must be a value in the range of 0 to 255. When <duration> is specified, the speaker sounds for an interval specified by <duration> multiplied by 0.1 second.

SAMPLE PROGRAMME

```
LIST

100 FOR I=1 TO 68
110 READ A,B
120 SOUND A,B
130 NEXT
140 END
200 DATA 8,8,8,4,9,4,10,8,10,4,11,4,12,8
,13,4,12,4,10,16
210 DATA 12,8,11,4,10,4,9,16,11,8,10,4,9
,4,8,16
220 DATA 8,8,8,4,9,4,10,8,10,4,11,4,12,8
,13,4,12,4,10,16
230 DATA 12,8,11,4,10,4,9,8,10,4,9,4,8,3
2
240 DATA 12,8,11,4,10,4,9,16,11,8,10,4,9
,4,8,16
250 DATA 12,8,11,4,10,4,9,16,11,8,10,4,9
,4,8,16
260 DATA 8,8,8,4,9,4,10,8,10,4,11,4,12,8
,13,4,12,4,10,16
270 DATA 12,8,11,4,10,4,9,8,10,4,9,4,8,2
4
>
```

*STAT

FORMAT STAT [| ALL |]
| <expression> |

PURPOSE To display the status of each programme area.

EXAMPLE STAT 3

REMARKS STAT displays the name and size of a programme stored in each programme area.

<expression> is the programme area number (1 to 5). If <expression> is used, the data in the area specified by the programme area number is displayed. If omitted, the data in the current programme area is displayed. If ALL is specified, the data from all the programme areas are displayed simultaneously along with the size of the RAM file area, MEMSET addresses, and the size of the unused text area (bytes free). However, as the size of the unused text area displayed here includes a work area for BASIC programme execution, it does not necessarily mean that all the unused text area is available for programme storage.

(See MEMSET and TITLE.)

STOP

FORMAT **STOP**
PURPOSE To terminate programme execution and return to command level.
EXAMPLE **STOP**

REMARKS STOP statements may be used anywhere in a programme to terminate execution and to return BASIC to command level. When a STOP statement is executed, the following message is output:

Break in nnnn (where nnnn is a line number at which the STOP statement has been executed.)

Unlike the END statement, the STOP statement does not close files. When BASIC has returned to command level, programme execution can be resumed by a CONT command.

(See CONT and END.)

SWAP

FORMAT **SWAP <variable 1>,<variable 2>**
PURPOSE To exchange the values of two variables.
EXAMPLE **SWAP A$, B$**

REMARKS With a SWAP statement, any type variable (integer, single precision, double precision, or string) may be swapped, but the two variables must be of the same type or a TM ("Type mismatch") error will occur.
When the type of <variable 2> is a simple variable and the value of the variable is not assigned, an FC ("Illegal function call") error will occur.

SAMPLE PROGRAMME

```
LIST

100 A=123.456
110 B=999999
120 PRINT "A=";A
130 PRINT "B=";B
140 SWAP A,B
150 PRINT
160 PRINT "SWAP!!"
170 PRINT
180 PRINT "A=";A
190 PRINT "B=";B
200 END
RUN
A= 123.456
B= 999999

SWAP!!

A= 999999
B= 123.456
>
```

*TITLE

FORMAT **TITLE <programme name>**
PURPOSE To name programmes.
EXAMPLE **TITLE"TEST 1"**

REMARKS TITLE names a programme in the programme area currently LOGged IN. <programme name> may be 8 characters max. Once assigned, this name will appear on the menu when a STAT command is executed or when power is applied to the HX-20.

The programme named by the TITLE statement can be executed directly by menu selection immediately after the power application to the HX-20, and is protected against accidental erasure by a NEW or LOAD command. If you attempt to execute either command in a programme area named by TITLE, a PP ("Protected programme") error will occur.

(See STAT.)

TRON/TROFF

FORMAT TRON
TROFF

PURPOSE To trace the execution of programme statements.

EXAMPLE

REMARKS Execution of a TRON statement (in either the direct or indirect mode) as an aid in debugging enables a trace flag that displays each line number of the programme as it is executed. The numbers appear enclosed in square brackets. The trace flag is disabled when a TROFF statement or NEW command is executed.

SAMPLE PROGRAMME

```
LIST

10 FOR I=1 TO 5
20 PRINT I
30 NEXT
40 END
TRON
RUN
[10][20] 1
[30][20] 2
[30][20] 3
[30][20] 4
[30][20] 5
[30][40]
TROFF
RUN
 1
 2
 3
 4
 5
>
```

*WIDTH

FORMAT **WIDTH <characters per line>, <number of lines> [,<scroll margin>]**

PURPOSE To set the size of the virtual screen

EXAMPLE **WIDTH 20, 25, 5**

REMARKS WIDTH specifies the size of the virtual screen by <characters per line> and <number of lines>. <characters per line> and <number of lines> can be set arbitrarily within the ranges of 20 to 255 and 4 to 255, respectively.
The size of the virtual screen is limited by the capacity of the memory. If you set the screen width too large, an OM ("Out of memory") error may occur. The minimum values are the size of the LCD screen.

<scroll margin> is an allowance for margins that you can provide on the left and right of the display screen, so that you can read the display screen easily when it is moved laterally.
When the cursor comes to the edge of the screen, the next character is outside the display screen and will therefore not be visible. For this reason, when the cursor has moved over a certain set width, it is advisable to move the screen also while the next character is still in view. In this way, the character to the left of the cursor is always on display.
When the cursor has moved without leaving the scroll margin, it indicates that the cursor has reached the end of the virtual screen.

At warm start, the size of the virtual screen is set at WIDTH 40, 8, 3 for the LCD and WIDTH 40, 37, 5 for the external display.
When a WIDTH command is executed, all files opened at that time are closed and all variables, character areas and data defined by DEF statements are cleared.
You may set the size of the virtual screen freely.
When you perform programme corrections, the larger the virtual screen, the more easily you can use it. However, with a very large screen, you may not be able to find the statements at all. Even if statements are being output correctly on the virtual screen, there may be no visible changes in the display screen.

*WIDTH <device name>

FORMAT WIDTH ["LPT0:" / "COM0:"], <number of digits>

PURPOSE To set the print width of the printer.

EXAMPLE WIDTH "LPT0:", 20

REMARKS WIDTH sets the maximum value of characters per line that can be output on the printer. Normally, for PRINT statements (including LPRINT and PRINT#) LINE FEED (LF) signal is sent to the printer by a semicolon at the end of each such statement. By specifying the print width with a WIDTH command, LF signal is automatically sent to the printer when <number of digits> which specifies the number of characters to be printed per line is reached. <number of digits> must have a value in the range of 1 to 255. However, 255 is interpreted as infinite and no automatic line feed according to the specified print width will take place.

NOTE:

At warm start, <number of digits> is set to 80 for the RS-232C port ("COM0:"). Therefore, if a long string exceeding 80 characters is output, line feed is effected automatically upon output of 80 characters. This will also be the result if a device other than the printer is connected to the RS-232C port of the HX-20. For this reason, when using the RS-232C port for communication with an external device, a string longer than 80 characters may not be transmitted properly with <number of digits> set at the default value. When programmes are to be transferred from one HX-20 to another HX-20, first execute WIDTH "COM0:", 255 to disable the automatic line feed operation and then start the transmission.

WIND

FORMAT **WIND [<counter value>]**
PURPOSE To control the microcassette drive for fastforward and rewind.
EXAMPLE **WIND 0**

REMARKS WIND causes the microcassette tape to be fast forwarded or rewound according to <counter value>. <counter value> must be within the range of −32768 to 32767.
The counter value of the microcassette recorder can be read by TAPCNT function.
Once this value is stored in memory, the target file can be searched irrespective of the current tape position by instructing the value with a WIND command.
If <counter value> is omitted, the microcassette recorder rewinds the tape to its beginning and resets the counter value to 0.
When a file is to be searched using a WIND command, you must execute in advance a WIND without <counter value>.

(See TAPCNT.)

SAMPLE PROGRAMME

```
LIST

100 COUNT=TAPCNT
110 OPEN "O",#1,"CAS0:TEST.DAT"
120 FOR A=1 TO 10
130 PRINT#1,A;SQR(A)
140 NEXT
150 CLOSE #1
160 WIND COUNT
170 OPEN "I",#1,"CAS0:TEST.DAT"
175 PRINT :PRINT" A ","SQR(A)"
180 IF EOF(1) GOTO 220
190 INPUT#1,A,B
200 PRINT A,B
210 GOTO 180
220 CLOSE #1
230 END
RUN

 A             SQR(A)
 1              1
 2              1.41421
 3              1.73205
 4              2
 5              2.23607
 6              2.44949
 7              2.64575
 8              2.82843
 9              3
 10             3.16228
>
```

CHAPTER 4
Functions

ABS

FORMAT ABS(<numeric expression>)
PURPOSE To return the absolute value of a numeric expression.
EXAMPLE A=ABS(−1.6)

REMARKS ABS returns the absolute value of <numeric expression>.

ASC

FORMAT ASC(<string>)
PURPOSE To return the character code of a character.
EXAMPLE A=ASC("A")

REMARKS ASC returns a numerical value that is the ASCII code of the first character of <string>. See APPENDIX F, "Character Code Table" for the relationship between characters and ASCII character codes.
If <string> is null, an FC ("Illegal function call") error is returned.

(See CHR$, and APPENDIX F, "Character Code Table".)

ATN

FORMAT ATN(<numeric expression>)
PURPOSE To return the arc tangent of a numeric expression.
EXAMPLE A=ATN(0.5)

REMARKS ATN returns the arc tangent of <numeric expression> in radians. The result of the operation is in the range $-\pi/2$ to $\pi/2$.

CDBL

FORMAT CDBL(<numeric expression>)
PURPOSE To convert integers and single precision numbers into double precision numbers.
EXAMPLE A#=CDBL(B!/2)

REMARKS CDBL converts the value of <numeric expression> to a double precision number. Only the type conversion is performed and there is no change in the number of significant digits.

CHR$

FORMAT **CHR$(<numeric expression>)**
PURPOSE To return the character corresponding to a specified character code.
EXAMPLE **A$=CHR$(&H41)**

REMARKS CHR$ returns the ASCII character whose code is the value of <numeric expression>. If the value of <numeric expression> is not in the range of 0 to 255, an FC ("Illegal function call") error occurs.
You may include real numbers in <numeric expression> but the fractional portion of the real number must be rounded before using it as the value of <numeric expression>.

(See ASC, and APPENDIX F, "Character Code Table.")

CINT

FORMAT **CINT(<numeric expression>)**
PURPOSE To convert single and double precision numbers into integers.
EXAMPLE **A%=CINT(B#/2)**

REMARKS CINT converts the value of <numeric expression> into an integer by rounding the fractional portion. If the value of <numeric expression> is not in the range of −32768 to 32767, an OV ("Overflow") error occurs.

(See CDBL, FIX, and INT.)

COS

FORMAT COS (<numeric expression>)
PURPOSE To return the cosine of a numeric expression.
EXAMPLE A=COS(3.1415926/2)

REMARKS COS returns the cosine of <numeric expression> in radians.

CSNG

FORMAT CSNG(<numeric expression>)
PURPOSE To convert integers and double precision numbers into single precision numbers.
EXAMPLE A!=CSNG(B#)

REMARKS CSNG converts the value of <numeric expression> into a single precision number in 6 significant digits.
If the value of <numeric expression> is not in the range −1.70141 E+38 to 1.70141 E+38, an OV ("Overflow") error occurs.

(See CDBL and CINT.)

CSRLIN

FORMAT CSRLIN
PURPOSE To return the vertical position of the cursor on the virtual screen.
EXAMPLE Y=CSRLIN

REMARKS CSRLIN returns the value of the vertical position of the cursor on the virtual screen. The value of the vertical position must be in the range of 0 to (number of lines on the virtual screen −1).

(See POS.)

DATE$

FORMAT DATE$ [=MM/DD/YY]
PURPOSE To set the current date in, and return the date kept by, the internal calendar clock.
EXAMPLE PRINT DATE$

REMARKS DATE$ displays the date kept by the internal calendar clock in the HX-20. Using this statement, you can set the date in the form of a string such as MM/DD/YY (e.g., "08/15/82") where MM represents the month, DD represents the day and YY represents the year.
The date is displayed in the same format as that when it was input. Once you set the correct current date with a DATE$, you are not required to set it again, as the clock in the HX-20 keeps track of the time and date.

(See DAY and TIME$.)

DAY

FORMAT DAY
PURPOSE To set the current day of the week in, and display the day of the week kept by, the internal calendar clock.
EXAMPLE PRINT DAY

REMARKS In the HX-20, the day of the week is kept by the internal calendar clock by integers 1 to 7 corresponding to the 7 days of the week. Using this statement, you can set the current day of the week such as DAY=7(Saturday). The day of the week is displayed by one of integers 1 to 7. Once you set the correct current day of the week with a DAY, you are not required to set it again.

(See DATE$ and TIME$.)

EOF

FORMAT EOF(<file number>)
PURPOSE To return the end-of-file code.
EXAMPLE IF EOF(3) 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.

ERL/ERR

FORMAT ERL
ERR

PURPOSE To return the error code of an error occurred and the line number where the error occurred.

EXAMPLE
```
A=ERL
B=ERR
```

REMARKS When an error occurs, the error code is stored in the variable ERR and the line number where the error occurred is stored in the variable ERL. If the statement that caused the error was executed in the direct mode, line number 65535 is stored in the variable ERL.
Normally, the ERL and ERR variables are used in the error trapping routine specified by an ON ERROR GOTO statement to control the processing flow.

(See ON ERROR GOTO.)

EXP

FORMAT EXP (<numeric expression>)

PURPOSE To return the value of an exponential function with e as its base.

EXAMPLE
```
A=EXP(1)
```

REMARKS The value of <numeric expression> must be the result of an exponential function. If the value of <numeric expression> is greater than 88.02969, an OV ("Overflow") error occurs.

FIX

FORMAT FIX (<numeric expression>)
PURPOSE To return the truncated integer part of a numeric expression.
EXAMPLE A=FIX(−B/3)

REMARKS FIX returns the value of <numeric expression> as a truncated integer part.

(See CINT and INT.)

FRE

FORMAT FRE(<expression>)
PURPOSE To return the size of an unused memory area.
EXAMPLE PRINT FRE(0)
PRINT FRE("A$")

REMARKS If the <expression> is a numeric expression, FRE returns the number of free bytes in the BASIC text area. If <expression> is a string expression, FRE returns the number of bytes in the BASIC string area. <expression> is merely a dummy. Any arguments to FRE may be assigned as long as they are numeric or string expressions. Since the size of the unused memory area displayed includes a work area for BASIC programme execution, please use the displayed memory capacity as a guide only.

HEX$

FORMAT HEX$(<numeric expression>)

PURPOSE To return a string which represents the hexadecimal value of the decimal argument.

EXAMPLE A$=HEX$(65535)

REMARKS HEX$ converts the decimal value of <numeric expression> to a hexadecimal value and returns it as a string. The value of <numeric expression> must be in the range −32768 to 65535. If the value of <numeric expression> includes a decimal fraction, it is rounded to an integer before HEX$ (<numeric expression>) is evaluated.

(See OCT$ and VAL.)

SAMPLE PROGRAMME

```
LIST

100 PRINT " DEC   OCT HEX"
110 FOR I=5 TO 16
120 PRINT USING"####";I;
130 PRINT USING"&  &";" ",OCT$(I),HEX$(I
)
140 NEXT I
RUN
 DEC   OCT HEX
   5     5   5
   6     6   6
   7     7   7
   8    10   8
   9    11   9
  10    12   A
  11    13   B
  12    14   C
  13    15   D
  14    16   E
  15    17   F
  16    20   10
>
```

INKEY$

FORMAT INKEY$

PURPOSE To return a one-character string of the pressed character key or a null string if no character key is pressed.

EXAMPLE A$=INKEY$

REMARKS INKEY$ returns a null string if the keyboard buffer is empty. If the keyboard buffer contains any character key input, INKEY$ reads the character from the buffer and returns it as a one-character string. Any keys not included in the "Character Code Table" such as **SHIFT** key, etc., are ignored.

(See APPENDIX F, "Character Code Table".)

INPUT$

FORMAT **INPUT$(<number of characters>[,[#]<file number>])**
PURPOSE To return a string of characters read from a specified file.
EXAMPLE **A$=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 echoed (i.e., 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.

INSTR

FORMAT INSTR([<numeric expression>,]<string 1>, <string 2>)

PURPOSE To search for the first occurrence of one string in another string and return the position of the searched string.

EXAMPLE B=INSTR(A$,"XYZ")

REMARKS INSTR searches for the first occurrence of <string 2> in <string 1> and returns the position at which the match is found. If <string 2> cannot be found, INSTR returns 0.
<numeric expression> is the position for starting the search. If <numeric expression> is omitted, the search is started from the beginning of <string 1>. If a null string is specified as <string 2>, INSTR returns the same value as that specified by <numeric expression>.

SAMPLE PROGRAMME

```
LIST

100 A$="ZXCVBNM,./AS FG JKL :"
110 B$=INPUT$(1)
120 C=INSTR(A$,B$)
130 IF C=0 GOTO 110
140 IF C>10 THENC=C+17
150 SOUND C+5,2
160 GOTO110
>
```

INT

FORMAT INT(<numeric expression>)
PURPOSE To return the largest integer value (truncated).
EXAMPLE PRINT INT (−B/3)

REMARKS INT returns the largest integer value which is equal to or less than the value of <numeric expression>.

(See FIX and CINT.)

SAMPLE PROGRAMME

```
LIST

100 PRINT "    I      CINT    INT     FIX"
110 FOR I=2.4 TO -2.4 STEP -0.3
120 A%=CINT(I)
130 B%=INT(I)
140 C%=FIX(I)
150 PRINT USING "  ##.#";I;
160 PRINT USING "  #### ";A%;B%;C%
170 NEXT I
180 END
RUN
    I      CINT    INT     FIX
   2.4      2       2       2
   2.1      2       2       2
   1.8      2       1       1
   1.5      2       1       1
   1.2      1       1       1
   0.9      1       0       0
   0.6      1       0       0
   0.3      0       0       0
   0.0      0       0       0
  -0.3      0      -1       0
  -0.6     -1      -1       0
  -0.9     -1      -1       0
  -1.2     -1      -2      -1
  -1.5     -1      -2      -1
  -1.8     -2      -2      -1
  -2.1     -2      -3      -2
  -2.4     -2      -3      -2
>
```

LEFT$

FORMAT LEFT$(<string>,<numeric expression>)
PURPOSE To return an arbitrary length of string from the leftmost characters of a string.
EXAMPLE B$=LEFT$(A$,4)

REMARKS The value of <numeric expression> must be in the range of 0 to 255. If <numeric expression> is greater than the total number of characters in <string>, the entire string will be returned. If <numeric expression> is 0, the null string (length zero) is returned.

(See MID$ and RIGHT$.)

LEN

FORMAT LEN(<string>)
PURPOSE To return the total number of characters in a string.
EXAMPLE A=LEN(A$)

REMARKS LEN returns the total number of characters in <string>.
If <string> includes non-printing characters such as control codes and blanks, they are not actually output but are counted as characters.

(See APPENDIX F, Character Code Table.)

LOF

FORMAT LOF (<file number>)
PURPOSE To return the size of a specified file.
EXAMPLE A=LOF(3)

REMARKS The file specified by <file number> must have been opened in the input mode. If the specified file number is in a ROM cartridge, LOF returns the remaining length of the file in units of bytes. If the specified file number is in the RS-232C port, LOF returns the number of data stored in the buffer in units of bytes.

LOG

FORMAT LOG(<numeric expression>)
PURPOSE To return the natural logarithm of a numeric expression.
EXAMPLE PRINT LOG(2.7812818)

REMARKS LOG returns the natural logarithm of the value given by <numeric expression>.

MID$

FORMAT MID$(<string>,<expression 1>[,<expression 2>])
PURPOSE To return an arbitrary length of string from a string.
EXAMPLE B$=MID$(A$,2,3)

REMARKS MID$ returns a string of characters in the length specified by <expression 2> from the <string> beginning with the <expression 1>th character. The values of <expression 1> and <expression 2> must be in the ranges 1 to 255 and 0 to 255, respectively.
If <expression 2> is omitted or if there are fewer than <expression 2> characters to the right of the <expression 1>th character, all rightmost characters beginning with the <expression 1>th character are returned. When the number of characters in <string> is fewer than <expression 1>, MID$ returns a null string.

(See LEFT$ and RIGHT$.)

SAMPLE PROGRAMME

```
LIST

100 A$="ABCDEFGHIJKLMN"
110 PRINT "       MID$        LEFT$
RIGHT$ "
120 FOR I=1 TO 7
130 M$=MID$(A$,I,7)
140 L$=LEFT$(A$,I)
150 R$=RIGHT$(A$,I)
160 PRINT USING "    &        &";M$,L$,R$
170 NEXT I
RUN
      MID$         LEFT$         RIGHT$
    ABCDEFG       A             N
    BCDEFGH       AB            MN
    CDEFGHI       ABC           LMN
    DEFGHIJ       ABCD          KLMN
    EFGHIJK       ABCDE         JKLMN
    FGHIJKL       ABCDEF        IJKLMN
    GHIJKLM       ABCDEFG       HIJKLMN
>
```

OCT$

FORMAT **OCT$(<numeric expression>)**
PURPOSE To return a string which represents the octal value of the decimal argument.
EXAMPLE **PRINT OCT$(123+456)**

REMARKS OCT$ converts the decimal value of <numeric expression> to an octal value and returns it as a string. The value of <numeric expression> must be in the range of −32768 to 65535. If the value of <numeric expression> includes a decimal fraction, it is rounded to an integer before OCT$ (<numeric expression>) is evaluated.

(See HEX$.)

*PEEK

FORMAT **PEEK(<address>)**
PURPOSE To return the byte read from a specified memory location.
EXAMPLE **A=PEEK(&H0C00)**

REMARKS PEEK returns the byte read from the memory location specified by <address> <address> must be in the range of 0 to 65535 (&H0 to &HFFFF). If the value of <address> includes a decimal fraction, it is rounded to an integer.
Since the memory space from addresses &H0 to &H4D is a special area allocated for input/output, an overrun may occur simply by reading any of these addresses.
In EPSON BASIC, when a PEEK is executed for any of the abovementioned addresses, an FC ("Illegal function call") error occurs. To read these addresses, &H80 must be written into &H7E (to set MSB).

(See POKE.)

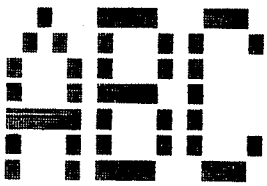
POINT

FORMAT **POINT(<horizontal coordinate>,<vertical coordinate>)**
PURPOSE To return the status of a dot at a specified location on the graphic screen.
EXAMPLE **PRINT POINT(100,10)**

REMARKS POINT checks whether or not a dot has been drawn at the location specified by <horizontal coordinate> and <vertical coordinate> on the graphic screen. With the LCD, POINT returns 1 if a dot has been set at the specified location and 0 if no dot has been set. With the external display, color codes 0 to 3 are returned. Please note that the LCD is different from the external display in the range within which you can specify graphic coordinates.

SAMPLE PROGRAMME

```
100 CLS:DEFINT A-Z
110 DIM P(17,7)
120 PRINT "ABC"
130 FOR Y=0 TO 7
140 FOR X=0 TO 17
150 P(X,Y)=POINT(X,Y)
160 NEXT X,Y
170 FOR Y=0 TO 7
180 PRINT
190 FOR X=0 TO 17
200 IF P(X,Y) THEN A$="■
" ELSE A$=" "
210 PRINT A$;
220 NEXT X,Y
```



POS

FORMAT POS(<digit>)

PURPOSE To return the horizontal position of the cursor on the virtual screen or the horizontal position of the printer head.

EXAMPLE X=POS(0)

REMARKS The value of <digit> must be in the range of 0 to 16. If 0 is specified, POS returns the horizontal position of the cursor on the virtual screen.
If any of the integers from 1 to 16 is specified, POS returns the number of characters stored in the buffer (namely, the number of characters that has been output following the output of the LF code and before execution of this function) on the file opened using the integer as a file number.
These numbers also correspond to the horizontal positions of the printer head.

(See CSRLIN and LOF.)

RIGHT$

FORMAT RIGHT$(<string>,<numeric expression>)

PURPOSE To return an arbitrary length of string from the rightmost characters of a string.

EXAMPLE PRINT RIGHT$("ABCD",3)

REMARKS The value of <numeric expression> must be in the range of 0 to 255. If <numeric expression> is greater than the total number of characters in <string>, the entire string will be returned. If <numeric expression> is 0, the null string (length zero) is returned.

(See LEFT$ and MID$.)

RND

FORMAT RND[(<numeric expression>)]
PURPOSE To return a random number.
EXAMPLE A=RND(1)

REMARKS RND returns a random number between 0 and 1. The random number generated varies with the value of <numeric expression> as follows.

- If <numeric expression> is negative, a new sequence of random numbers is generated.
- If <numeric expression> is 0, the last generated random number is repeated.
- If <numeric expression> is positive, the next random number in the sequence is generated.

If <numeric expression> is omitted, the next random number in the sequence is also generated. The same sequence of random numbers is generated each time a RUN or CLEAR statement is executed, if the random number generator is not reseeded by a RANDOMIZE statement.

SAMPLE PROGRAMME

(See RANDOMIZE.)

```
LIST

100 DEFINT N,B
110 DIM B(120)
120 INPUT "NUMBER OF REP
ETITIONS ";A
130 CLS
140 FOR I=0 TO A
150 N=RND*120

160 B(N)=B(N)+1
170 PSET (N,31-B(N))
180 NEXT

>
RUN

NUMBER OF REPETITION
S ?  2000
```



SGN

FORMAT **SGN(<numeric expression>)**
PURPOSE To return the sign of the value of a numeric expression.
EXAMPLE **B=SGN(A)**

REMARKS If the value of <numeric expression> is positive, SGN returns 1. If the value of <numeric expression> is 0, SGN returns 0. If the value of <numeric expression> is negative, SGN returns −1.

SIN

FORMAT **SIN(<numeric expression>)**
PURPOSE To return the sine of a numeric expression.
EXAMPLE **PRINT SIN(3.1415926/2)**

REMARKS SIN returns the sine of <numeric expression> in radians.

(See COS and TAN.)

SPACE$

FORMAT SPACE$(<numeric expression>)
PURPOSE To return a string of spaces of a specified length.
EXAMPLE A$="A"+SPACE$(10)+"C"

REMARKS SPACE$ returns a string of spaces of the length specified by <numeric expression>. The value of <numeric expression> must be in the range of 0 to 255.

(See SPC and TAB.)

SPC

FORMAT SPC(<digit>)
PURPOSE To output a specified number of blanks.
EXAMPLE PRINT SPC(10);"A"

REMARKS SPC outputs blanks in the number specified by <digit>. This function may only be used in output statements such as PRINT. The value of <digit> must be in the range of 0 to 255.

(See SPACE$ and TAB.)

SQR

FORMAT SQR(<numeric expression>)
PURPOSE To return the square root of a numeric expression.
EXAMPLE A=SQR(2)

REMARKS SQR returns the square root of <numeric expression>.
The value of <numeric expression> must be greater than 0.

STR$

FORMAT STR$(<numeric expression>)
PURPOSE To return a string representation of the value of a numeric expression.
EXAMPLE A$=STR$(123)

REMARKS STR$ converts the value specified by <numeric expression> to a string. For <numeric expression>, you can use any type of numeric constants.

(See STRING$ and VAL.)

STRING$

FORMAT STRING$ (<integer expression>, | <string expression> | <numeric expression> |)

PURPOSE To return a string of specified characters.

EXAMPLE PRINT STRING$(10,65)

REMARKS STRING$ returns a string of characters specified by <string expression> or <numeric expression> in the length specified by <integer expression>. If <string expression> is specified as a string of characters to be returned, all characters having the first character of the string are returned.
If <numeric expression> is specified, all characters having ASCII code specified by the numeric expression are returned. The value of <numeric expression> must be in the range of 0 to 255.

(See STR$.)

TAB

FORMAT **TAB(<numeric expression>)**
PURPOSE To space to a specified position on the line where the cursor is currently positioned.
EXAMPLE **PRINT TAB(10);"ABC"**

REMARKS TAB is used only in output statements such as PRINT and LPRINT.
TAB outputs blanks from the current cursor position to the position specified by <numeric expression> counted from the left-hand end on the virtual screen.
The value of <numeric expression> may be in the range of 0 to 255 with 0 as the leftmost position. In other words, value of <numeric expression> corresponds to the remainder when it is divided by the number of characters to be displayed horizontally. If the position specified by <numeric expression> is at the left of the current cursor position, TAB goes to that position on the next line.
Please note the difference between the SPC function and the TAB function.

(See SPC.)

TAN

FORMAT **TAN(<numeric expression>)**
PURPOSE To return the tangent of a numeric expression.
EXAMPLE **A=TAN(3.1416/4)**

REMARKS TAN returns the tangent of the value of <numeric expression> in radians.

(See COS and SIN.)

*TAPCNT

FORMAT TAPCNT
PURPOSE To return the value of the microcassette drive counter.
EXAMPLE PRINT TAPCNT
A=TAPCNT

REMARKS TAPCNT function is used to read the value of the microcassette drive counter. The returned value is in the range of −32768 to 32767. This counter value is always returned as positive after the counter has been reset by a WIND command. By assigning a value to TAPCNT, you can set the counter value.

(See WIND.)

SAMPLE PROGRAMME

```
LIST

100 TAPCNT=0
110 OPEN "O",#1,"CAS0:TEST"
120 FOR I=1 TO 10
130 PRINT#1,I;I*I
140 NEXT
150 CLOSE
160 WIND 0
170 OPEN "I",#1,"CAS0:TEST"
180 IF EOF(1) THEN 220
190 INPUT#1,A,B
200 PRINT A,B
210 GOTO 180
220 CLOSE
230 END
RUN
 1             1
 2             4
 3             9
 4             16
 5             25
 6             36
 7             49
 8             64
 9             81
 10            100
>
```

TIME$

FORMAT TIME$="HH:MM:SS"
PURPOSE To return the time kept by the internal calendar clock.
EXAMPLE PRINT TIME$

REMARKS TIME$ displays or sets the time kept by the internal clock of the HX-20. The time is set or displayed in the format "HH:MM:SS" where the value of HH ranges from 00 to 23 and the values of MM and SS range from 00 to 59. Once the correct time has been set, you are not required to set the time again, as the HX-20 clock keeps track of the time and date.

(See DATE$ and DAY.)

USR

FORMAT USR[<digit>](<argument>)
PURPOSE To call machine language subroutine defined by DEFUSR statement.
EXAMPLE A=USR 1(B)

REMARKS USR calls your machine language subroutine (user-defined function) with <argument>. Before calling the user-defined function, it must have been written into the memory, and its execution starting address must have been defined by the DEFUSR statement.
A maximum of 10 user-defined functions can be set by <digit> whose value is in the range of 0 to 9 and corresponds to the digit supplied with the DEFUSR statement for that routine. If <digit> is omitted, USR0 is assumed.
With <argument>, you can transfer a value from EPSON BASIC to your machine language subroutine.

(See DEF USR and Chapter 5, "Machine Language Subroutines".)

VAL

FORMAT VAL(<string expression>)
PURPOSE To return the numerical value of a string expression.
EXAMPLE A=VAL("−123")

REMARKS If the first character of <string expression> is not +, −, &, ., or a digit, VAL (<string expression>) = 0.
If a character other than digits (0 to F in hexadecimal numbers and 0 to 7 in octal numbers) appears, the following characters are ignored. Blanks in <string expression> are also ignored.

(See CHR$ and STR$.)

VARPTR

FORMAT VARPTR(<variable name>)
PURPOSE To return the address of a variable or array.
EXAMPLE PRINT HEX$(VARPTR(A))

REMARKS Any type of <variable name> (numeric, string, or array) can be specified. A value must be assigned to the variable specified by <variable name> before executing VARPTR.
The returned address will be an integer in the range of −32768 to 32767. If a negative address is returned, obtain the actual value by adding it to 65536.
Whenever a value is assigned to a new variable other than arrays, the addresses of the arrays change. Therefore, all simple variables must be assigned before calling VARPTR for an array.

(See Chapter 5, "Machine Language Programmes.")

CHAPTER 5
Additional Information

5.1 RAM files

RAM (random) files are one of the many features of EPSON BASIC.

With RAM files,
- Data can be accessed randomly.
- Data can be changed freely.
- High access speed is assured as compared with the I/O transfer speed to and from peripheral devices.

Because of the above advantages, RAM files may be regarded the same as array variables. RAM files also feature the following:

- Data is retained even after the power switch is turned OFF.
- Data can be shared by plural programmes.

In view of the above features, RAM files are ideal for data to be handled more frequently than those in sequential files, and are thus useful for:
- Storage of variable tables, conversion tables, etc., to be used constantly. (Scientific calculations)
- Storage of data necessary for daily transaction processing (Business management)

In EPSON BASIC, five separate programmes can be stored simultaneously. However, RAM files must be shared by these five programmes and cannot be used at the discretion of plural programmes. In using PUT% statments, be careful so that data files may not be accidentally destroyed by other programmes.

In fact, RAM files as a whole occupy a single area, but the RAM file area may be used as separate files apparently using a DEFFIL statement. The DEFFIL statement specifies the location of the first record in the RAM file area and the locations of records to be read or written by a GET% or PUT% statement after the execution of the DEFFIL will be shifted relatively.

The type of variable of the data to be read by a GET% statement must match that of the data written by a PUT% statement. The number of bytes occupied on the memory space by each type of variable is different as follows.

Integer variable	2 bytes
Single precision variable	4 bytes
Double precision variable	8 bytes
String variable	Indefinite

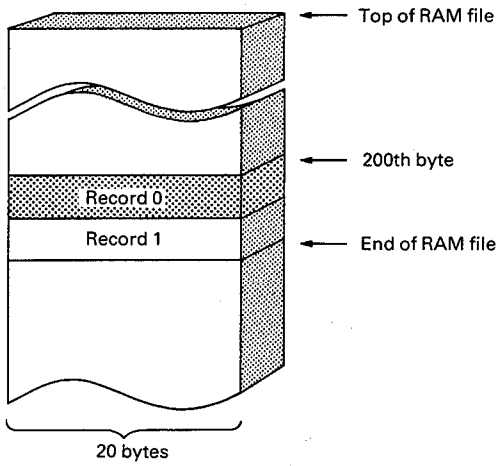
For this reason, if the data written as an integer value is to be read as a double precision or single precision value, the result will be an entirely different value.

The length of one record is determined by a DEFFIL statement and within that record, numeric variables occupy the length specified for each variable type. The remaining length of the record is allocated to string variables. This is because of the fact that the length of a string variable is indefinite, and in GET% and PUT% statements, string variables can be used only after numeric variables.

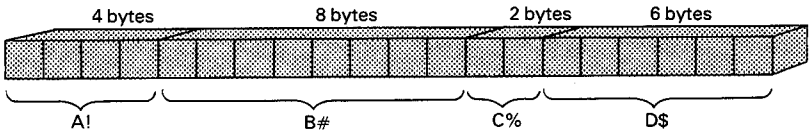
Example:

```
100  DEFFIL 20, 200
 :
200  PUT% 0, A!, B#, C%, D$
 :
300  GET% 0, E!, F#, G%, H$
 :
400  END
```

In this programme, the location of a file is as shown below.



The record length of, for example, Record 0 is allocated as follows.



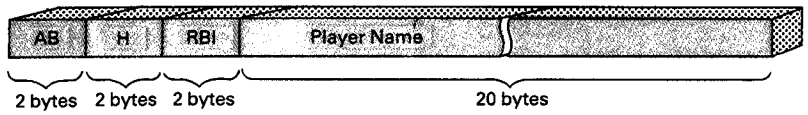
Execution of the programme results as follows.

```
E!=A!, F#=B#
G%=C%, H$=D$
```

As an example of the use of a RAM file, let's prepare a file to record the individual statistics for each of the players of a baseball team, using the player's uniform numbers as file numbers to reference the files.

The first thing you must consider is the record length for each player. In this example, let's deal with the At-Bats, Hits and RBI (Runs Batted In) for each player. You can handle all three values as integer values. You will need to record the player names in alphabetic characters. To this end, let's have a string variable of a maximum of 20 bytes.

AB (At-bats)	2 bytes
H (Hits)	2 bytes
RBI (Runs Batted In)	2 bytes
Player Name	20 bytes
Total	26 bytes



2 bytes 2 bytes 2 bytes 20 bytes

Next, you must determine the total file length required for the team's statistics. You already know the record length for each player. All that you must do here is to multiply the length of the individual record by the number of players on the team. Assuming that the team has 25 players, the total file length will be as follows.

$26 \times 25 = 650$ bytes

You can now write programmes for recording the individual player's statistics according to the required functions.

5.1.1 Creation of a RAM file

Though somewhat unusual, let's limit the player's uniform numbers to the range 0 to 24. In this programme, you only need to write the AB, H and RBI data available for each player up to this point.

```
100 CLEAR 200,650
110 DEFINT U,A,H,R
120 DEFFIL 26,0
130 INPUT "UNIFORM NO. "
;U
140 IF U>24 THEN 210
150 INPUT "NAME     ";NA$
160 INPUT "AT BATS ";A
170 INPUT "HITS     ";H
180 INPUT "RBI      ";R
190 PUT% U,A,H,R,NA$
200 GOTO 130
210 END
```

Even if you input a player name longer than 20 characters, no error will occur. However, as only 20 bytes are available for the player name field on a record in the RAM file, only the first 20 characters of the name will be stored. Since each of the three variables (AB, H and RBI) requires 2 bytes, the variables are declared as integer type by a DEFFINT statement.

5.1.2 Retrieval and updating of data in RAM file

This programme is used to read the data written into the RAM file by the programme in the preceding example. Since each player name is already entered in the file, all that is necessary for you to retrieve the data relating to a player is to input the player uniform number. If you type "Y" following the displayed data, the individual record can be updated. The newly entered statistics will then be added to the statistics currently recorded.

```
220 CLEAR 200,650
230 DEFINT U,A,H,R
240 DEFFIL 26,0
250 PRINT
260 INPUT "UNIFORM NO. "
;U
270 IF U>25 THEN END
280 GET% U,A,H,R,NA$
290 AV=H/CSNG(A)
300 PRINT NA$
310 PRINT "AT BATS ";A
320 PRINT TAB(10);"HITS
";H
330 PRINT "RBI ";R
340 PRINT TAB(10);"AVERA
GE ";AV
350 PRINT "UPDATE (Y/N)"
;
360 IF INPUT$(1)<>"Y" GO
TO 250
370 PRINT
380 INPUT "AT BATS ";XA
390 INPUT "HITS    ";XH
400 INPUT "RBI     ";XR
410 PUT% U,A+AX,H+HX,R+R
X
420 GOTO 280
```

In addition to the two sample programmes given, you can also write a programme to list the players' performance records in the order of batting average or a programme to select and list the individual records of, say, top 5 players. With these programmes and data files prepared by the HX-20, you can manage your baseball team practically.

5.2 Sequential files

If you consider the recording of music on an audio cassette, you will realize that musical numbers are recorded in sequence and that to listen to a particular musical number, you must advance the tape to the section of the tape where that number is recorded. A sequential file operates on the similar principle except that, instead of music, a sequence of data is recorded on the tape.

The speed and flexibility of RAM files are very attractive, but their memory capacity limits the amount of data that can be handled. For this reason, much data cannot be handled on each RAM file.
Also, when using a RAM file, you must write a programme by taking the internal file structure into account.

In contrast, sequential files feature ease of operation in addition to:

- Unlimited data handling capacity
- Simple file structure

However, despite such advantages, sequential files have the following drawbacks as well.

- Data can only be accessed in sequence.
- Data can therefore not be retrieved quickly.
- Partial changes to the data file cannot be effected.

When you prepare a programme using a RAM or sequential file, the relative merits and demerits of both types of files must be weighed before making your selection.
To use a sequential file, you must observe the following procedure.

(1) Open the file.
(2) Input data from the file to memory using an INPUT# statement, or output data from memory to the file using a PRINT# statement.
(3) Close the file.

Each of sequential files are assigned a file number when they are created and all subsequent I/O operations to and from these files (using INPUT#, PRINT#, etc.) are performed by specifying a particular file with this file number. Therefore, no two files can exist at the same time under the same file number. Also, once a file has been OPENed under a certain file number, no other files can be opened using that number until the file first opened is closed.

Data to be read with an INPUT# statement must match the data written with a PRINT# statement in both the variable type and the number of variables. In a sequential file, data is read in sequence from the beginning of the file. Therefore, if there is any difference in the number of variables, subsequent INPUT# statement will read all the data in incorrect sequence. If there is a difference in the type of variable, a TM ("Type Mismatch") error will occur.

After the I/O transfer using the sequential file, you must always execute a CLOSE (or END) statement. With EPSON BASIC, data transfer between the HX-20 and peripheral equipment is performed in units of 256 bytes. For example, when writing data into the file, a write operation will not take place until 256 bytes of data have accumulated in the buffer. If a CLOSE (or END) statement is not executed at the end of the programme, data will remain in the buffer and not be output. In this case, the file will also be left incomplete without a delimiter being written at the end of the file.

As an example of the use of a sequential file, let's write a programme for an address directory. In the preceding example of RAM file, remember, you had to carefully consider the record structure such as record description items and record length. However, with sequential files, as there are no particular restrictions on the amount of data, you are only required to decide on the record description items and the order in which you wish to file them.

5.2.1 Creation of a sequential file

This programme is to record a list of names, addresses, phone numbers and birth dates on an audio cassette tape.
Although you can write such data as name, address, etc., all together as a string, it is better to record by delimiting these description items for subsequent data utilization.

```
100 OPEN"O",#1,"CAS1:ADR
S"
110 PRINT
120 INPUT "NAME ";NAME$
130 IF NAME$="" THEN 250
140 INPUT "ADDRESS";ADR$
150 INPUT "TELEPHONE ";T
EL$
160 PRINT"DATE OF BIRTH"
170 INPUT "  YEAR";Y
180 INPUT " MONTH";M
190 INPUT "   DAY";D
200 PRINT#1,NAME$
210 PRINT#1,ADRS$
220 PRINT#1,TEL$
230 PRINT#1,Y;M;D
240 GOTO 110
250 CLOSE
260 END
```

If you observe the operation of the audio cassette, you will probably notice that the tape does not advance all the time. This is because of the action of the buffer described in the preceding section. That is, a write operation is performed only after a specified amount of data has accumulated in the buffer. After you input all the data for the required number of persons, press only the **RETURN** key at line 120 to assign a null string to A$. This will branch the programme to line 250 to CLOSE the file. This step cannot be skipped.
If you press **BREAK** key to terminate the programme after all the data input has been completed, data will remain in the buffer and the file will not be closed.

5.2.2 Retrieval of data from sequential file

To demonstrate the advantages of using this computer, rather than simply outputting the recorded data as it was entered, let's write a more useful programme to output selected data from the file, for example, the data of only those people born on a certain year.

```
100 OPEN "I",#1,"CAS1:AD
RS"
110 PRINT "PEOPLE BORN W
HAT YEAR?"
120 INPUT N
130 LPRINT " *** BORN IN
";N;"***"
140 IF EOF(1) THEN GOTO
230
150 INPUT#1,NAME$,ADR$,T
EL$,Y,M,D
160 IF N<>Y GOTO 140
170 LPRINT
180 LPRINT NAME$
190 LPRINT ADR$
200 LLISTNT TEL$
210 LPRINT "BORN ";M;"/"
;D;"/";Y
220 GOTO 140
230 LPRINT
240 LPRINT " *** END OF
RETRIEVAL *** "
250 CLOSE
260 END
```

In the preceding file creation programme, you have indicated the end of file by pressing RETURN key. However, when reading a file, this determination is made by the EOF function. If you omit line 140 in the above programme, an IE ("Input past end") error will occur.

5.2.3 Correction of data in sequential files

In a sequential file, you cannot change only a portion of the data recorded on a file. To correct the file, you must prepare another file and write correct data into the new file while reading the old data file. As the HX-20 allows you to use both a microcassette drive ("CAS0:") and an audio cassette ("CAS1:"), you can perform the file updating utilising these two units.

```
100 CNT=TAPCNT
110 OPEN "I",#1,"CASI:AD
RS"
120 OPEN "O",#2,"CASO:WO
RK"
130 IF EOF(1) GOTO 320
140 INPUT #1,NAME$,ADR$,
TEL$,Y,M,D
150 PRINT NAME$
160 PRINT ADR$
170 PRINT TEL$
180 PRINT "BORN ";M;"/";
D;"/";Y
190 PRINT "DO YOU WISH T
O MAKE CORRECTIONS (Y/N)
 ?"
200 IF INPUT$(1)<>"Y" GO
TO 290
210 PRINT
220 INPUT "NAME      ";NA
ME$
230 INPUT "ADDRESS   ";AD
R$
240 INPUT "TELEPHONE";TE
L$
250 PRINT "DATE OF BIRTH
 "
260 INPUT "   YEAR";Y
270 INPUT " MONTH";M
280 INPUT "    DAY";D
290 PRINT#2,NAME$,ADR$,T
EL$
300 PRINT#2,Y;M;D
310 GOTO 130
320 CLOSE
330 WIND CNT
340 PRINT "PLEASE REWIND
 THE CASSETTE"
350 PRINT
360 PRINT "IF OK, PLEASE
 INPUT "Y""
370 IF INPUT$(1)<>"Y" GO
TO 360
380 OPEN "O",#1,"CAS1:AD
RS"
390 OPEN "I",#2,"CAS0:WO
RK"
400 IF EOF(2) GOTO 440
410 INPUT#1,NAME$,ADR$,T
EL$,Y,M,D
420 PRINT#2,NAME$,ADR$,T
EL$,Y,M,D
430 GOTO 400
440 CLOSE
450 END
```

The actual transfer of data from the old to the new file terminates at line 320. In this state, the new file is in the microcassette drive. For subsequent data file utilisation, you must transfer the data in the new file back to the file in the audio cassette. The tape in the microcassette drive can be rewound using a WIND statement in a programme. The audio cassette, however, must be operated manually. In this sample programme, two files are used so that a larger amount of data may be stored.
If the amount of data to handle is small and all the data can be stored in the memory at one time, corrections of data file can be performed just the same as the RAM file.

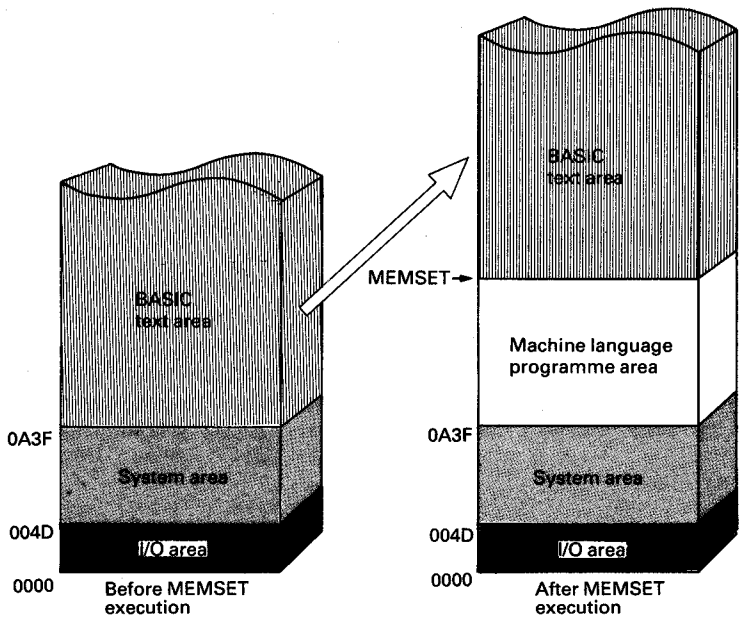
5.3 Machine language programmes

In EPSON BASIC, the USR function and the EXEC statement are provided as functions to call user programmes written in machine language. USR, like other functions, can pass data using an argument. The EXEC statement, however, has no function to pass any variables. In BASIC, even if a programme containing a bug is executed, execution stops but the programme itself will not be destroyed. However, in a machine language programme, even if a single bit is incorrect, all the programmes (including BASIC programmes) may be destroyed.

5.3.1 Memory allocation

When using user programmes written in machine language, memory space must be secured so that BASIC programmes and data are protected and that machine language programmes are not destroyed by any BASIC programmes. In EPSON BASIC, machine language programmes are placed before the BASIC programme area and the memory space for machine language programmes is secured using a MEMSET command. Therefore, you must execute this MEMSET command before loading any machine language programme.

The execution of the MEMSET command secures the machine language programme area and at the same time moves BASIC programmes. Therefore, BASIC programmes previously stored in memory are protected against destruction.



5.3.2 Writing and loading programmes

A machine language programme is loaded into the memory using the MONITOR function of the HX-20. Short programmes can be written into the memory using a POKE statement. A machine language programme you have written can also be stored as a machine language programme file in the same manner as BASIC programmes using a SAVEM or LOADM command.
The HX-20 has a vacant location for an expansion ROM socket to enable you to store your completely debugged programmes in the expansion ROM, so that you can always use them as the utility software of the HX-20. You can also store less frequently used programmes in the ROM cassette as a programme file.
In this way, programme loading time can be reduced greatly and the operability of the HX-20 can be improved as well.

5.3.3 USR function

The format of the USR function to call a machine language programme is as follows.

USR[(<digit>)](<argument>)

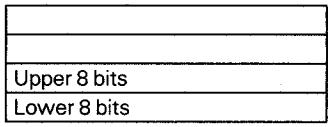
<digit> may be an integer from 0 to 9 and must correspond to the digit supplied in the DEFUSR statement. <argument> may be any numeric or string expression.
The USR function has one argument. The accumulator A contains a value that specifies the argument type.

The values used and their meanings are as follows.

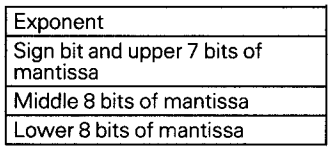
Value in accumulator	Argument type
2	Integer
3	String
4	Single-precision real number
8	Double-precision real number

If the argument type is numeric, the value in the index register shows the address of the "floating-point accumulator" where the argument is stored.
This floating-point accumulator does not refer to the memory location where a variable itself is stored, but refers to a special area used when BASIC performs an arithmetic operation. The actual value is stored in the "floating-point accumulator" in a different form depending on the argument type as follows.

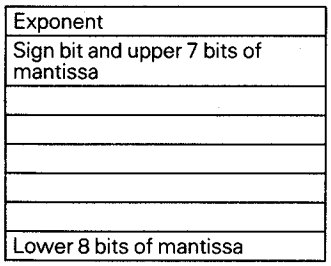
- When the argument is an integer:



- When the argument is a single-precision real number:

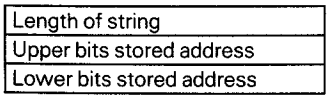


- When the argument is a double-precision real number:



- When the argument is a string:

  The value of the index register indicates the address of 3-byte data called "string descriptor".



The string descriptor is a pointer which indicates the address where a string type data is actually stored. By referring to the address indicated by this string descriptor, you can find the data passed as an argument.

When the USR function returns a value to BASIC, data must be stored in the same format in the location where the argument was stored (i.e., the location indicated by the floating-point accumulator or string descriptor). Therefore, the type of value returned from the USR function must be the same as that of the value specified as the argument.

To return to a BASIC programme from the USR function, machine language subroutine RTS (&H39) is used. For this reason, the value of the stack pointer when programme control is returned to BASIC must be the same as that when the USR function was called.

5.3.4 EXEC statement

Machine language subroutines can also be executed by an EXEC statement.

EXEC[<address>]

When a machine language subroutine is loaded into memory using a LOADM command, or when the execution starting address has been specified by EXEC, <address> can be omitted.

EXEC only executes the programme from the specified address and has no function to pass an argument. When variables are to be passed between a BASIC programme and a machine language programme called by an EXEC statement, it is performed through the direct read and write of the variables stored in memory. You can do this in the following ways.

(1) To POKE and PEEK the specified addresses of the machine language area using the BASIC programme.
(2) To directly read and write data in the BASIC variable area using the machine language programme.

When you use method (2), you must check beforehand the addresses of the variables to be written or read using the VARPTR function.

5.3.5 VARPTR function

With the VARPTR function, you can check at which address in memory your specified variable is stored. VARPTR is used to pass a variable to a machine language programme to be called by an EXEC statement, or to pass two or more variables using the USR function.

When using the VARPTR function, a value must have been assigned to the variable specified as the argument before execution of VARPTR.

The value returned by the VARPTR function is the top address of the specified variable data. If the variable is a string, the data is not the value of the variable but is a <string descriptor> which is a pointer indicating the address where the string type data is stored.

When a variable is stored in memory, the variable type, length of variable name and variable name (maximum 16 characters) are stored immediately before the value of the variable.

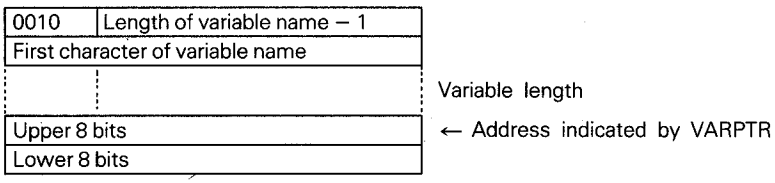
(1) Variable type
Using the first 4 bits of the first byte of data, variables are classified according to the length that the data occupies, as follows.

2 : Integer
3 : String
4 : Single-precision real number
8 : Double-precision real number

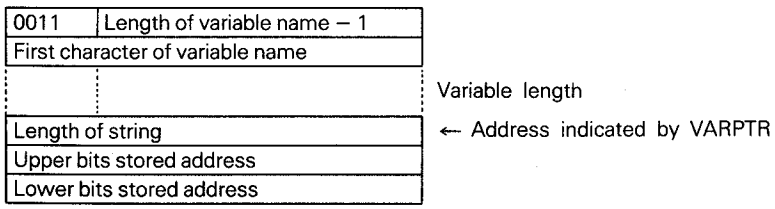
(2) Length of variable name
Using the last 4 bits of the first byte of data, the actual length of the variable name minus one is stored. Therefore, when the variable name is a single character, 0 is stored as the value.

(3) Variable name
Starting from the second byte of data, the variable name of the length indicated in (2) above is stored in ASCII format. Although the variable name is stored in a maximum of 16 characters, only the required number of bytes are used.

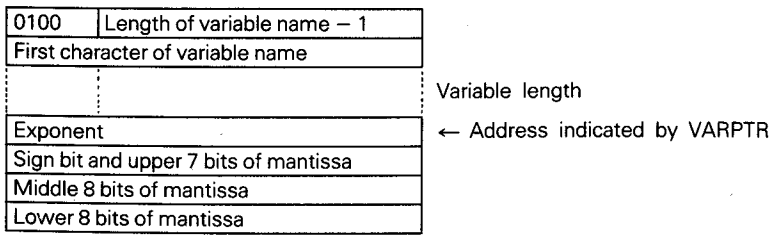
- When the variable type is an integer:



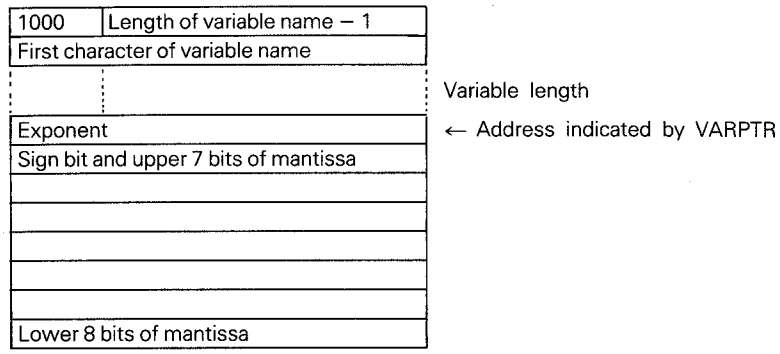
- When the variable type is a string:



- When the variable type is a single-precision real number:



- When the variable type is a double-precision real number:



Real-numbers type data are always normalized to omit the most significant bit of the mantissa. Also, whatever the value of the mantissa may be, if the exponent is 0, the value of the mantissa is assumed as 0. Compared to the index register which points the address of the floating-point accumulator when the USR function is called, the value of a variable specified by the VARPTR function indicates the address where the variable is stored. Do not confuse these two types of functions. Be especially careful with integer variables, as they are stored somewhat differently from other variable types.

5.4 How to use the RS-232C port

The HX-20 incorporates an RS-232C interface to allow communication with external devices. The RS-232C port in the HX-20 is normally used to connect an external printer with the HX-20. RS-232C is an EIA Standard for the interface between a MODEM and the associated data terminal equipment.

With this RS-232C interface, the HX-20 can communicate directly with other computers. By connecting an acoustic coupler to the RS-232C interface, the HX-20 can also transfer data to and from remote locations via subscriber lines.

The RS-232C standard is widely in use by various kinds of communications equipment. However, as this standard merely prescribes electrical characteristics, use of signal lines, transmission procedures, etc., differ from one type of equipment to another. Therefore, when connecting an external device to the HX-20, you must carefully check the external device for agreement of interface conditions with those of the HX-20 so that proper communications can be established under the same interface conditions.

With the HX-20, this setting of interface conditions can be performed using BASIC. (See OPEN"COM0:"). Once the uniform interface conditions have been established between the HX-20 and the external device, all other procedures are the same as when you use normal files. If you specify <file number> in a programme statement, you can perform I/O operations using PRINT# and INPUT# statements. With the HX-20 in the direct mode, you can directly transfer programmes between the HX-20 and external devices using LIST"COM0:" and LOAD"COM0:" commands.

NOTE:
If you execute an output command by specifying "COM0:" as the device name when no external device is connected to the RS-232C port or when the connection between the HX-20 and the external device is faulty, the operation of the HX-20 may terminate. Should this happen, press **BREAK** *key, to return the HX-20 to command level.*
Also, when a long line of characters is transferred without specifying the print width for "COM0:", the default value will be in effect and automatic line feed will take place at every 80 characters. When you send a long string exceeding 80 characters, you must first execute WIDTH "COM0:", 255 to set the print width as infinite.

5.4.1 Interfacing with optional devices

All the devices available as the options to the HX-20 have standardised signal lines. Therefore, as long as the exclusive interface cables specified by EPSON are used, you are not required to have a special knowledge of the RS-232C interface.

(1) Interfacing between two HX-20 units

You must use interface cables (optional cable set #715) when interfacing your HX-20 with another HX-20 unit. In this method, you do not need to set the interface conditions and can omit <BLPSC>.
NOTE:
When directly transferring programmes using a LIST "COM0:" or LOAD "COM0:" command, data may be lost during the data transfer if the data processing speed of the receiving HX-20 cannot catch up with the data transfer speed of the transmitting equipment. To transfer programmes properly in such a case, be sure to lower the bit rate of the transmitting equipment.

(2) Interfacing with a terminal printer

When connecting an external terminal printer to the HX-20, the printer must be equipped with an RS-232C interface. (For any of EPSON MX series printers, use interface board #8141 or #8145.) You must also use special interface cables (cable set #714).
There are two methods of using a terminal printer depending on whether you use the interface conditions of the HX-20 or those of the terminal printer.

- When setting the interface conditions at the terminal printer:

You need not set the interface conditions <BLPSC> for your HX-20. By simply specifying <device name> as "COM0:" in either direct mode or programme mode, you can use the terminal printer just the same as the built-in microprinter "LPT0:".

Set the interface conditions of the terminal printer as follows. (For details, see the user's manual of the applicable interface board.)

Bit rate	4,800 bps
Word length	8 bits
Parity bit	No parity
Stop bit length	2 bits

- When setting the interface conditions at the HX-20:

  If you specify <device name> as "COM0:", you must set the interface conditions <BLPSC> as follows.

Bit rate	
Word length	Set to the same parameters
Parity bit	of the terminal printer.
Stop bit length	
Control lines active	Specify "B" (hex) when using an EPSON MX series interface board.

Example:

Bit rate	300 bps
Word length	7 bits
Parity bit	Even parity
Stop bit length	1 bit
Control lines active	Handshaking by DSR signal (To match to the factory-set conditions interface board #8145, specify "COM0"(27E1B)".)

NOTE:
The maximum bit rate of the HX-20 is 4,800 bps. If the bit rate of the printer has been set at 9,600 bps, you must change the setting of the printer to 4,800 bps, as you cannot make this parameter compatible with that of the printer at the HX-20.

(3) Interfacing with an acoustic coupler
To interface the HX-20 with the optional acoustic coupler CX-20, you must use special interface cables (optional cable set #706).
When using the acoustic coupler, the interface conditions of both the HX-20 and CX-20 must be set to match to those of the transmitting/receiving equipment.

Word length Parity bit Stop bit length	Set to the same parameters of the transmitting/receiving equipment.
Bit rate	Must be the same as that of the transmitting/receiving equipment but must be 300 bps max. due to limitations of telephone lines. ($0 \leqslant B \leqslant 2$)
Control lines active	As all control lines are normally used, specify "2" (hex).

5.4.2 Interfacing with other external equipment

To interface the HX-20 with external devices other than those specified as the options to the HX-20, you must have a deep knowledge of the RS-232C interface. You can set the bit rate, word length, parity bit and stop bit length to match to those of the device to be connected to the HX-20. But the signal lines used, the method of handshaking, etc. are different from one device to another.

Simply connecting the signal lines of the same name will not assure successful communication between the two devices. You must have correct understanding of the function of each signal line.

The HX-20 uses the following 9 signal lines.

Pin No.	Signal	Signal direction	Function
1	GND	–	Signal GND
2	TXD	Out	Transmitted data
3	RXD	In	Received data
4	RTS	Out	Request to send
5	CTS	In	Clear to send
6	DSR	In	Data set ready
7	DTR	Out	Data terminal ready
8	CD	In	Carrier detect
E	FG	–	Protective GND

- Pin No. 1 – GND
  Signal of zero potential which serves as reference for all other signals.
- Pin No. 2 – TXD
  Output signal line to send data from the HX-20. Data is output in negative logic.
- Pin No. 3 – RXD
  Input signal line to receive data by the HX-20. Data is input in negative logic.
- Pin No. 4 – RTS
  Output signal line to request the grant for data transmission. When this signal (normally of positive potential) is active, it indicates that the HX-20 is ready for data transmission. By specifying "C" in <BLPSC> setting, it can be changed to that a signal of negative potential has significance.

- Pin No. 5 – CTS

  Input signal line to receive the grant for data transmission. If this signal line is at positive potential, the HX-20 judges that the request for data transmission has been granted, and starts data transmission.
  If the potential of this signal changes to negative, the HX-20 stops transmission and waits until the signal potential becomes positive to restart transmission. By specifying "C" in <BLPSC> setting, this signal can be ignored. If it is done so, the signal potential is always assumed as positive. When interfacing with equipment other than the acoustic coupler, this signal is often directly connected to the RTS signal, but in the HX-20, data transmission and reception can also be controlled by this signal.

- Pin No. 6 – DSR

  Input signal line used by the HX-20 to find if the counterpart is ready to receive data. When the potential of this signal is positive, the HX-20 judges that the counterpart is ready to receive data. Therefore, data tranmission can be terminated by changing this signal potential to negative. This is the main signal used by the HX-20 to control the transmission of data. By specifying "C" in <BLPSC> setting, this signal can be ignored. If this is done, the potential of this signal is always assumed as positive. This signal is originally intended to function as a response signal to the DTR signal to detect whether or not the communication circuit is ready. However, in the HX-20, this line is used as an input signal line to detect the BUSY state of the transmitting/receiving peripheral. Therefore during data transmission, the HX-20 always monitors this control line.

- Pin No. 7 – DTR

  This signal line is intended to output a signal by the HX-20 to the peripheral devices to request information as to whether the communication circuits are ready or not. This information request is made when the level of this signal is positive.
  In the HX-20, this signal line has another function. When the HX-20 is to function as a receiving equipment, this output signal informs the transmitting equipment of whether or not the HX-20 is ready to receive data. When the potential of this signal is high-impedence, the HX-20 is not ready to receive data, that is, BUSY.
  NOTE:
  DTR is connected to the power supply line of the RS-232C and high-speed serial interface driver. For this reason, if the RS-232C interface or high-speed serial interface is turned ON, DTR signal will automatically be activated.

- Pin No. 8 – CD

  Input signal line to detect that data is being sent from the transmitting equipment to the HX-20. When the potential of this signal is positive, the HX-20 judges that data is to be sent from the peripheral connected to the HX-20. This control line is required when the HX-20 is connected to the acoustic coupler. In all other cases, this signal can be ignored by specifying "C" in <BLPSC> setting.

In the HX-20, data transmission and reception can be controlled by signal lines CTS and DSR. You can find the status of data reception by signal line DTR. When you connect the HX-20 to any other external device, you must check carefully the functions of all the signal lines provided in the device and make required connections based on the function and not the nomenclature of each line.

APPENDIXES

APPENDIX A Error Messages

/0 11 Division by zero

A division by zero is encountered in an expression.

- The divisor in an expression is zero.
- Division by an undefined variable is encountered in an expression.
- The argument of TAN function is $\pi/2$.

AO 52 File already open

A file of the specified number is already open.

- A file opened has not been closed in direct mode.

BD 58 Bad data in file

Data format in a file is incorrect.

- An attempt is made to read a programme file in binary format as a data file in ASCII format.
- An attempt is made to read a machine language programme file as a data file.

BF 51 Bad file mode

A file mode is incorrect.

- An attempt is made to open a file with a file mode not allowed for that device (e.g., open the printer for the input mode, etc.).
- An attempt is made to execute an I/O command that is inconsistent with the mode in which the file was opened (e.g., output data to a file opened for the input mode, etc.).

BN 50 Bad file number

A file number is incorrect.

- A file number not in the range 1 to 16 is used in an OPEN statement.
- A file number not in the range 1 to 16 is used in an I/O statement.

BO 61 Buffer overflow

An overflow occurred in the input buffer.

- Data input from the RS-232C port ("COM0:") is overflowed. (In data transfer without handshaking, the bit rate is too fast. The bit rate should be reduced.)

BS 9 Bad subscript

A subscript that is outside the dimensions of the array, or the wrong number of subscripts is used.

- The size of array variable elements in a DIM statement is too large. (Normally, this will result in an OM or OV error.)
- The value of a subscript other than that declared in a DIM statement is used.
- The number of dimensions for an array is incorrect.

- A subscript with a value greater than 11 is used without declaration by a DIM statement.
- A subscript specified as 0 is referenced after execution of an OPTION BASE 1.
- A record number in a PUT% or GET% statement is too large.

CN 17 Can't continue

Programme execution cannot be resumed.

- A programme has halted due to an error.
- The programme has been modified after it was BREAKed.
- A programme is not executed.
- An abort has occurred, as **BREAK** key was pressed during I/O operation.

DD 10 Duplicate definition

An array or user function is defined in duplication.

- An array of the same name is declared without executing an ERASE statement.
- Undeclared array variables are used and then declared by a DIM statement.
- Attempts to execute an OPTION BASE statement was made twice.
- A DIM or DEF FN statement exists in a loop.

DS 56 Direct statement in file

During a LOAD or MERGE operation, an unnumbered programme line is read.

- A data file is read.
- A machine language programme is read.

DU 60 Device unavailable

A device is not available.

- A device which is not connected to the HX-20 is specified.

FC 5 Illegal function call

A statement or function is called incorrectly.

- A parameter for a statement or function is out of range. (Many functions cannot be used with a negative or zero parameter.)
- The value of a subscript in an array is negative.
- An undefined USR function is used.
- The number of characters specified in a PRINT USING statement exceeds 25.
- An undeclared array or a variable to which no value has been assigned is used in a SWAP statement.
- A line number greater than 64000 is encountered during execution of a RENUM command.
- In a PCOPY command, the specified area is the programme area currently LOGged IN, or a programme already exists in the specified area, or no programme exists in the programme area currently LOGged IN.
- Offset of a DEFFIL statement is too large.
- PEEK or POKE is executed against the EPSON BASIC programme area or the I/O area up to address &H4D.

FD 55 Bad file descriptor
A file descripter is incorrect.
- An element of the file descriptor is misspelled.

FN 23 FOR without NEXT
NEXT statements are insufficient.
- One NEXT statement is shared by two or more FOR statements.
- FOR and NEXT do not correspond one to one. (The control variable name is written incorrectly.)

ID 12 Illegal direct
A statement that is illegal in direct mode is entered.
- DEF FN, INPUT and RANDOMIZE statements, etc., cannot be executed in direct mode.

IE 54 Input past end
All the data in the file has been read.
- The number of data and the number of variables to be read do not match.
- An attempt is made to read data continuously without using the EOF function.

IO 53 Device I/O error
An error has occurred during communication with a peripheral device.
- A cassette tape is defective.
- The level adjustments of the audio cassette are mismatched.
- The interface conditions of the RS-232C (bit rate, handshaking lines, etc.,) are mismatched.

IU 59 Device in use
The specified device is busy.
- Wrong device name.
- The same OPEN statement is executed twice.
- Execution of CLOSE statement is neglected.

LS 15 String too long
A string is too long.
- An attempt is made to assign a string variable longer than 256 characters.

MO 22 Missing operand
A required parameter is missing in an expression.
- A full stop is used instead of a comma between numbers.
- An essential parameter is omitted.

NE 63 File not exist

A file does not exist under the specified name.

- A file name is written incorrectly.
- A ROM cartridge that does not contain the specified file is used.

NF 1 NEXT without FOR

FOR statements are insufficient.

- Incorrect looping is executed.
- Two or more NEXT statements exist for one FOR statement.
- Accidental jump to FOR–NEXT loop from other programme line.

NO 57 File not OPEN

A file number is used for a file that has not been OPENed.

- A file number is written incorrectly.
- An OPEN statement is not programmed.

NR 19 No resume

No RESUME statement is contained in an error trapping routine.

- At the end of an error trapping routine, there must be one of the following statements: END, RESUME, and ON ERROR GOTO.

OD 4 Out of data

A READ statement is encountered when there is no data to read.

- The number of data is insufficient.
- A RESTORE statement is incorrect.
- Use of delimiters in a DATA statement is incorrect.

OM 7 Out of memory

Memory capacity is insufficient.

- A programme is too long.
- A programme has too many variables.
- An array variable is too large.
- Expressions are too complicated.
- A programme has too many FOR...NEXT or GOSUB...RETURN loops.
- The string space or RAM file size specified by a CLEAR command is too large.
- The address number specified by a MEMSET statement is too large.

OS 14 Out of string space

A string space is insufficient.

- The string space specified by a CLEAR command is too small.

OV 6 Overflow

The result of a calculation is too large.

- The result of an operation on integer constants is not in the range −32768 to 32767.
- The result of an operation on real numbers is not in the range −1.70141E38 to 1.70141E38.
- A value in a command using the address as a parameter exceeds the specified range.

PP 62 Protected programme

The programme is protected.

- A NEW or LOAD command cannot be executed for the programme in the area which has been named by a TITLE statement.

RG 3 RETURN without GOSUB

A RETURN statement is encountered before the execution of a GOSUB statement.

- Execution is branched to a subroutine by a GOTO statement.
- A subroutine is executed by a RUN command.
- In the absence of an END statement at the last line of the main programme the following subroutine is executed.

RW 20 RESUME without error

A RESUME statement is executed when no error exists.

- An error trapping routine is entered by a GOTO or GOSUB statement.
- In the absence of an END statement at the last line of the main programme, the following error trapping routine is executed.

SN 2 Syntax error

A programme is not written according to the syntax of the language used.

- A reserved word other than those recognised is used.
- Unmatched parentheses.
- A delimiter is mistyped (comma for full stop, colon for semicolon, etc.).
- A variable name does not start with an alphabetic character.
- A variable name starts with a reserved word.
- The number of parameters for a function or statement is incorrect.
- Unrelated characters are written in the latter part of a line not visible on the physical screen.
- A string variable is used before the variable name list in a PUT% or GET% statement.

ST 16 String formula too complex

A string expression is too complex.

- A string expression written in one line is too long or complex. Too many nested parentheses are used in a string expression.

TM 13 Type mismatch

A mismatch in the type of variable.

- A numeric value name is assigned to a string variable.
- A string value name is assigned to a numeric variable.
- A type mismatch exists in the argument of a function.

UF 18 Undefined user function

A USR function is not defined.

- A variable name starting with "FN" is used.
- The function name in a DEF FN statement is incorrect.
- The DEF FN statement is not executed. (Execution of a programme is started from the middle of the programme by a GOTO or similar statement.)

UL 8 Undefined line number

An error in the line number.

- Line number is not specified.
- A line number specified in a GOTO, GOSUB, RESTORE or RUN statement does not exist.
- The line to be referenced when a RENUM statement is executed does not exist.

UP 21 Unprintable error

Indicates an error with an undefined error code.

- An ERROR statement is executed in the absence of any error trapping routine.
- Error codes 26 to 49 and 64 to 255 will cause this message to be displayed.

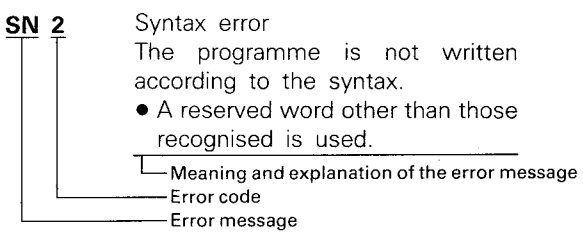
WE 24 WHILE without WEND

- This message is used in Disk BASIC.

WH 25 WEND without WHILE

- This message is used in Disk BASIC.

How to Read Error Message Table



APPENDIX B Device Names

Device name	Equipment name	Input	Output	Remarks
KYBD:	Keyboard	○	×	
SCRN:	Screen	×	○	
LPT0:	Built-in microprinter	×	○	
COM0:	RS-232C port	○	○	
CAS0:	Microcassette drive	○	○	Option
CAS1:	Audio cassette	○	○	
PAC0:	ROM cartridge	○	×	Option
A:	Flexible disk drive A	○	○	Device
B:	Flexible disk drive B	○	○	names for
C:	Flexible disk drive C	○	○	DISK BASIC
D:	Flexible disk drive D	○	○	

○: Applicable ×: Not applicable

APPENDIX C
Correspondence Table between Device Names and BASIC Commands

Command \ Device	KYBD:	SCRN:	LPT0:	COM0:	CAS0:	CAS1:	PAC0:
LOAD	×	×	×	○	○	○	○
LOADM	×	×	×	×	○	○	○
LOAD?	×	×	×	×	○	○	×
RUN "<file descriptor>"	×	×	×	○	○	○	○
MERGE	×	×	×	○	○	○	○
FILES	×	×	×	×	○	○	○
INPUT#	○	×	×	○	○	○	○
INPUT$	○	×	×	○	○	○	○
FOF	–	×	×	○	○	○	○
LOF	–	×	×	○	–	–	○
SAVE	×	○	○	○	○	○	×
SAVEM	×	×	×	×	○	○	×
LIST	×	○	○	○	○	○	×
PRINT# (USING)	×	○	○	○	○	○	×
POS	×	○	○	○	–	–	×
OPEN mode	I	O	O	I/O	I/O	I/O	I

NOTE:
○ *or* × *used in this table indicates that when a device is specified for a command or statement, the device*

○: Can be used.
×: Cannot be used. An FC error occurs.
–: Causes no error but the command is invalid.

APPENDIX D Formatting Characters

Format string	Function
!	Specifies to output only the first character in a given string.
\···\	Specifies to the number of characters to be output from the beginning of a given string.
&	Specifies the output positions of characters in a given string.
#	Specifies each digit position.
.	Specifies the position of the decimal point.
+	Outputs the sign of a number (plus or minus) before or after the number.
–	Outputs negative numbers with a trailing minus sign.
**	Causes leading spaces in the numeric field to be filled with asterisks.
$$	Causes a dollar sign to be output to the immediate left of the formatted number.
**$	Causes leading spaces to be filled with asterisks and a dollar sign to be output before the number.
,	Causes a comma to be output to the every 3rd digit to the left of the decimal point.
^^^^	Outputs a numeric value in exponential format.
_	Outputs any of the above formatting characters as a literal character.

NOTE: The formatting characters shown above apply to the ASCII character set. If your selected character set is other than ASCII, some of the formatting characters will be output differently as shown below.

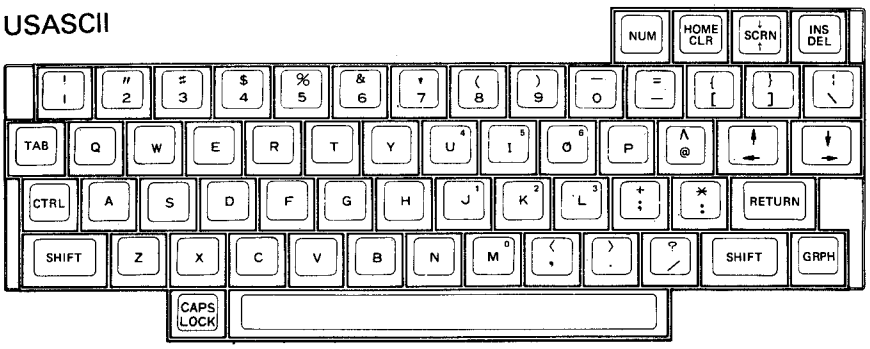
U.S.A.	France	Germany	England	Denmark	Sweden	Italy	Spain
#	#	#	£	#	#	#	Pt
$	$	$	$	$	¤	$	$
\	ç	Ö	\	ø	Ö	\	Ñ
^	^	^	^	^	Ü	^	^

(See Chapter 3 POKE).

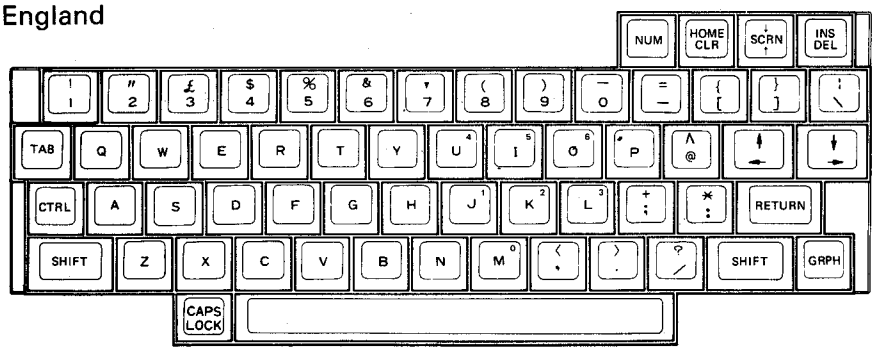
APPENDIX E
Keyboard Layouts and Key Assignments

1. Keyboard Layouts

USASCII



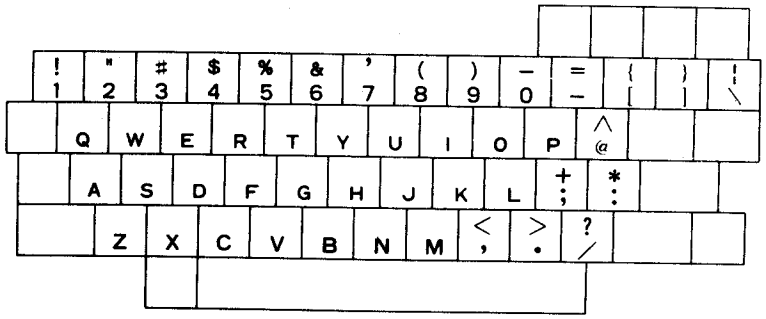
England



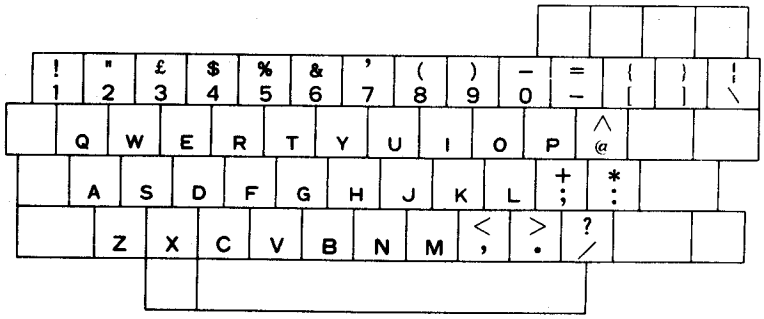
2. Key Assignments for Each Keyboard Mode

(1) Uppercase Mode and Lowercase Mode

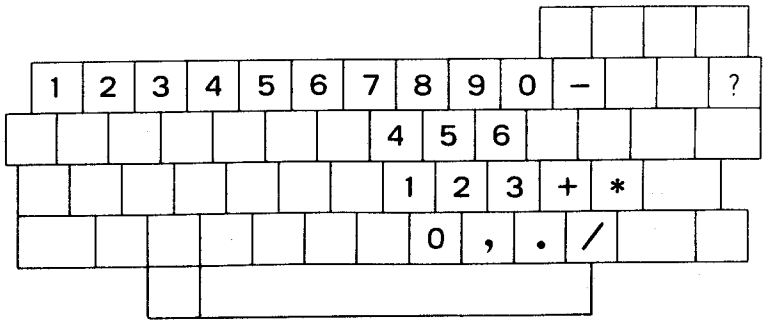
USASCII



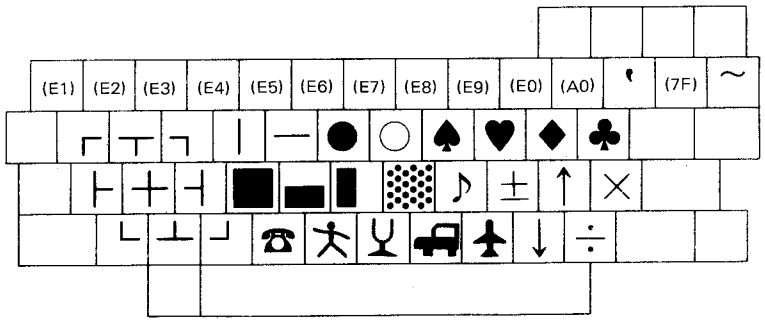
England



(2) Numeric Key Mode



(3) Graphic Key Locations

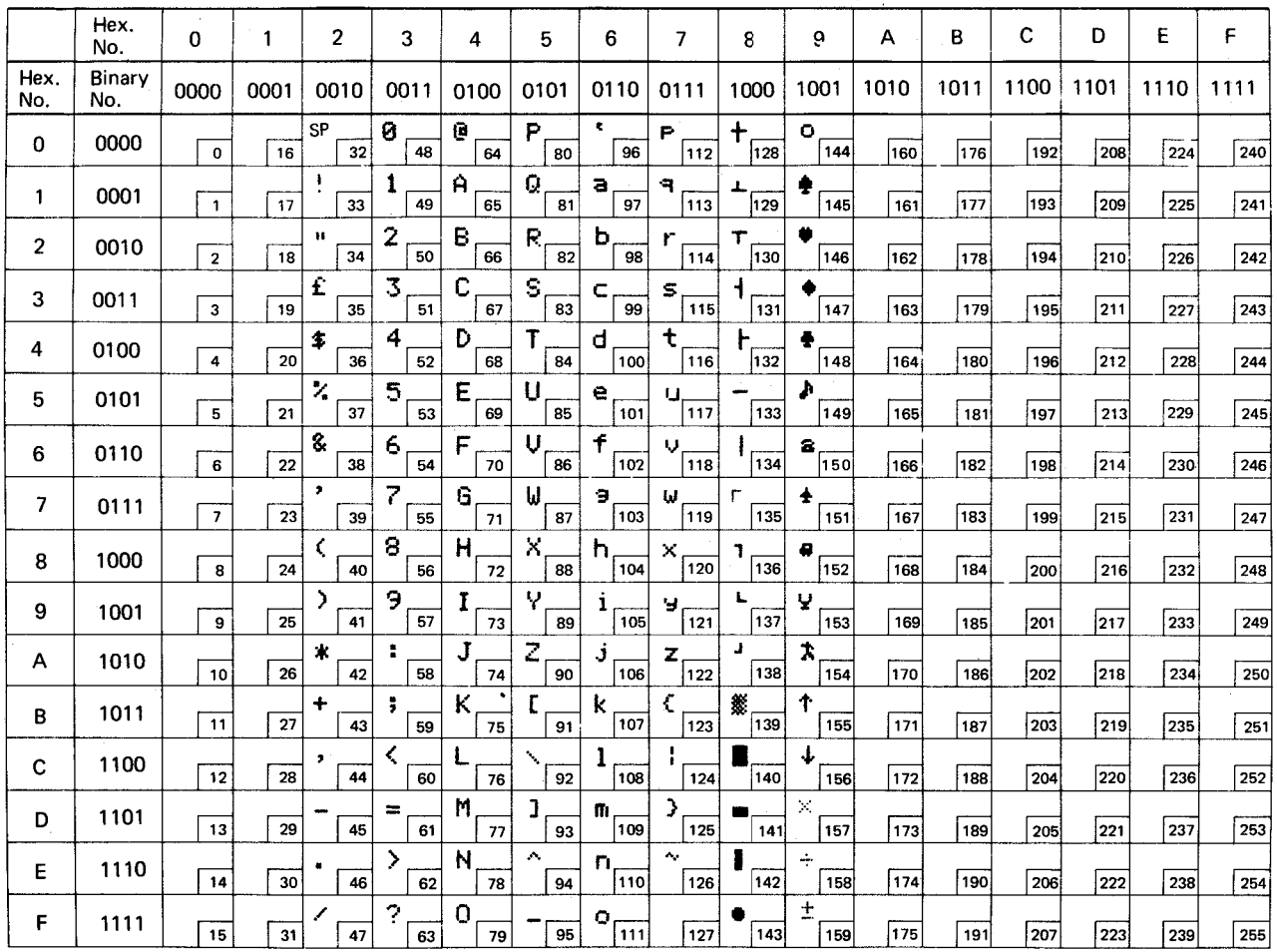


NOTE: E0 through E9, A0 and 7F shown in parentheses in the above figure are character codes in hexadecimal numbers and can be input by pressing the corresponding keys while holding down the GRPH key. The character codes for ‘ and ~ are 60 and 7E, respectively. (See Chapter 8, "Definition of Graphic Pattern" in the HX-20 Operation Manual.)

APPENDIX F Character Code Tables

1. USASCII

	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0 16		SP 32	0 48	@ 64	P 80	` 96	p 112	+ 128	○ 144	160	176	192	208	224	240
1	0001	1 17	! 33	1 49	A 65	Q 81	a 97	q 113	⊥ 129	♣ 145	161	177	193	209	225	241
2	0010	2 18	" 34	2 50	B 66	R 82	b 98	r 114	T 130	♠ 146	162	178	194	210	226	242
3	0011	3 19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	4 20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	5 21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6 22	& 38	6 54	F 70	V 86	f 102	v 118	┼ 134	♂ 150	166	182	198	214	230	246
7	0111	7 23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	♀ 151	167	183	199	215	231	247
8	1000	8 24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	♫ 152	168	184	200	216	232	248
9	1001	9 25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	☼ 153	169	185	201	217	233	249
A	1010	10 26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	→ 154	170	186	202	218	234	250
B	1011	11 27	+ 43	; 59	K 75	[91	k 107	{ 123	▒ 139	← 155	171	187	203	219	235	251
C	1100	12 28	, 44	< 60	L 76	\ 92	l 108	¦ 124	■ 140	↓ 156	172	188	204	220	236	252
D	1101	13 29	- 45	= 61	M 77	] 93	m 109	} 125	▪ 141	× 157	173	189	205	221	237	253
E	1110	14 30	. 46	> 62	N 78	^ 94	n 110	~ 126	▬ 142	÷ 158	174	190	206	222	238	254
F	1111	15 31	/ 47	? 63	O 79	_ 95	o 111	127	● 143	± 159	175	191	207	223	239	255



	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	@ 64	P 80	` 96	p 112	┼ 128	○ 144	160	176	192	208	224	240
1	0001	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♠ 145	161	177	193	209	225	241
2	0010	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	♥ 146	162	178	194	210	226	242
3	0011	19	£ 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	22	& 38	6 54	F 70	V 86	f 102	v 118	│ 134	ä 150	166	182	198	214	230	246
7	0111	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	à 151	167	183	199	215	231	247
8	1000	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	å 152	168	184	200	216	232	248
9	1001	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	ç 153	169	185	201	217	233	249
A	1010	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	ê 154	170	186	202	218	234	250
B	1011	27	+ 43	; 59	K 75	[91	k 107	{ 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	28	, 44	< 60	L 76	\ 92	l 108	¦ 124	■ 140	↓ 156	172	188	204	220	236	252
D	1101	29	- 45	= 61	M 77	] 93	m 109	} 125	▬ 141	× 157	173	189	205	221	237	253
E	1110	30	. 46	> 62	N 78	^ 94	n 110	~ 126	▌ 142	÷ 158	174	190	206	222	238	254
F	1111	31	/ 47	? 63	O 79	_ 95	o 111	127	● 143	± 159	175	191	207	223	239	255

	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	à 64	P 80	` 96	p 112	┼ 128	○ 144	160	176	192	208	224	240
1	0001	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♠ 145	161	177	193	209	225	241
2	0010	2	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	● 146	162	178	194	210	226	242
3	0011	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	◆ 147	163	179	195	211	227	243
4	0100	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6	22	& 38	6 54	F 70	V 86	f 102	v 118	│ 134	ä 150	166	182	198	214	230	246
7	0111	7	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	± 151	167	183	199	215	231	247
8	1000	8	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	■ 152	168	184	200	216	232	248
9	1001	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	Ü 153	169	185	201	217	233	249
A	1010	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	↑ 154	170	186	202	218	234	250
B	1011	11	27	+ 43	; 59	K 75	° 91	k 107	é 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	12	28	, 44	< 60	L 76	Ç 92	l 108	ù 124	▌ 140	↓ 156	172	188	204	220	236	252
D	1101	13	29	- 45	= 61	M 77	§ 93	m 109	è 125	▬ 141	× 157	173	189	205	221	237	253
E	1110	14	30	. 46	> 62	N 78	^ 94	n 110	¨ 126	▐ 142	÷ 158	174	190	206	222	238	254
F	1111	15	31	/ 47	? 63	O 79	_ 95	o 111	127	● 143	± 159	175	191	207	223	239	255

	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	§ 64	P 80	' 96	p 112	+ 128	○ 144	160	176	192	208	224	240
1	0001	17	! 33	1 49	A 65	Q 81	a 97	q 113	⊥ 129	♠ 145	161	177	193	209	225	241
2	0010	18	" 34	2 50	B 66	R 82	b 98	r 114	⊤ 130	♥ 146	162	178	194	210	226	242
3	0011	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	22	& 38	6 54	F 70	V 86	f 102	v 118	┃ 134	ä 150	166	182	198	214	230	246
7	0111	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	± 151	167	183	199	215	231	247
8	1000	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	ß 152	168	184	200	216	232	248
9	1001	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	Ö 153	169	185	201	217	233	249
A	1010	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	Ü 154	170	186	202	218	234	250
B	1011	27	+ 43	; 59	K 75	Ä 91	k 107	ä 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	28	, 44	< 60	L 76	Ö 92	l 108	ö 124	■ 140	↓ 156	172	188	204	220	236	252
D	1101	29	- 45	= 61	M 77	Ü 93	m 109	ü 125	▬ 141	× 157	173	189	205	221	237	253
E	1110	30	. 46	> 62	N 78	^ 94	n 110	ß 126	▌ 142	÷ 158	174	190	206	222	238	254
F	1111	31	/ 47	? 63	O 79	_ 95	o 111	127	● 143	± 159	175	191	207	223	239	255

	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	@ 64	P 80	` 96	p 112	+ 128	o 144	160	176	192	208	224	240
1	0001	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	⊥ 129	♠ 145	161	177	193	209	225	241
2	0010	2	18	" 34	2 50	B 66	R 82	b 98	r 114	⊤ 130	♥ 146	162	178	194	210	226	242
3	0011	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6	22	& 38	6 54	F 70	V 86	f 102	v 118	┼ 134	◘ 150	166	182	198	214	230	246
7	0111	7	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	◘ 151	167	183	199	215	231	247
8	1000	8	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	◘ 152	168	184	200	216	232	248
9	1001	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	◘ 153	169	185	201	217	233	249
A	1010	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	◘ 154	170	186	202	218	234	250
B	1011	11	27	+ 43	; 59	K 75	Æ 91	k 107	æ 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	12	28	, 44	< 60	L 76	ø 92	l 108	¢ 124	▌ 140	↓ 156	172	188	204	220	236	252
D	1101	13	29	- 45	= 61	M 77	Å 93	m 109	ä 125	▬ 141	→ 157	173	189	205	221	237	253
E	1110	14	30	. 46	> 62	N 78	^ 94	n 110	~ 126	▐ 142	← 158	174	190	206	222	238	254
F	1111	15	31	/ 47	? 63	O 79	_ 95	o 111	127	● 143	± 159	175	191	207	223	239	255

	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	Ø 48	É 64	P 80	é 96	p 112	┼ 128	○ 144	160	176	192	208	224	240
1	0001	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♠ 145	161	177	193	209	225	241
2	0010	2	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	♥ 146	162	178	194	210	226	242
3	0011	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	4	20	¤ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6	22	& 38	6 54	F 70	V 86	f 102	v 118	│ 134	◘ 150	166	182	198	214	230	246
7	0111	7	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	↑ 151	167	183	199	215	231	247
8	1000	8	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	◙ 152	168	184	200	216	232	248
9	1001	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	Ö 153	169	185	201	217	233	249
A	1010	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	Ü 154	170	186	202	218	234	250
B	1011	11	27	+ 43	; 59	K 75	Ä 91	k 107	ä 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	12	28	, 44	< 60	L 76	Ö 92	l 108	ö 124	▌ 140	↓ 156	172	188	204	220	236	252
D	1101	13	29	- 45	= 61	M 77	Å 93	m 109	å 125	▬ 141	→ 157	173	189	205	221	237	253
E	1110	14	30	. 46	> 62	N 78	Ü 94	n 110	ü 126	▐ 142	← 158	174	190	206	222	238	254
F	1111	15	31	/ 47	? 63	O 79	_ 95	o 111	127	▪ 143	± 159	175	191	207	223	239	255

7. ITALY

Hex. No. / Binary No.	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0	0	16	SP 32	0 48	@ 64	P 80	ù 96	p 112	╪ 128	○ 144	160	176	192	208	224	240
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♣ 145	161	177	193	209	225	241
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	♥ 146	162	178	194	210	226	242
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♠ 148	164	180	196	212	228	244
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118	┼ 134	ª 150	166	182	198	214	230	246
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119	╞ 135	↑ 151	167	183	199	215	231	247
8	8	24	(40	8 56	H 72	X 88	h 104	x 120	╘ 136	ø 152	168	184	200	216	232	248
9	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	↓ 153	169	185	201	217	233	249
A	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	Ü 154	170	186	202	218	234	250
B	11	27	+ 43	; 59	K 75	° 91	k 107	ä 123	▒ 139	¢ 155	171	187	203	219	235	251
C	12	28	, 44	< 60	L 76	\ 92	l 108	ö 124	▌ 140	£ 156	172	188	204	220	236	252
D	13	29	- 45	= 61	M 77	é 93	m 109	è 125	▐ 141	× 157	173	189	205	221	237	253
E	14	30	. 46	> 62	N 78	^ 94	n 110	ì 126	▀ 142	÷ 158	174	190	206	222	238	254
F	15	31	/ 47	? 63	O 79	_ 95	o 111	ß 127	● 143	± 159	175	191	207	223	239	255

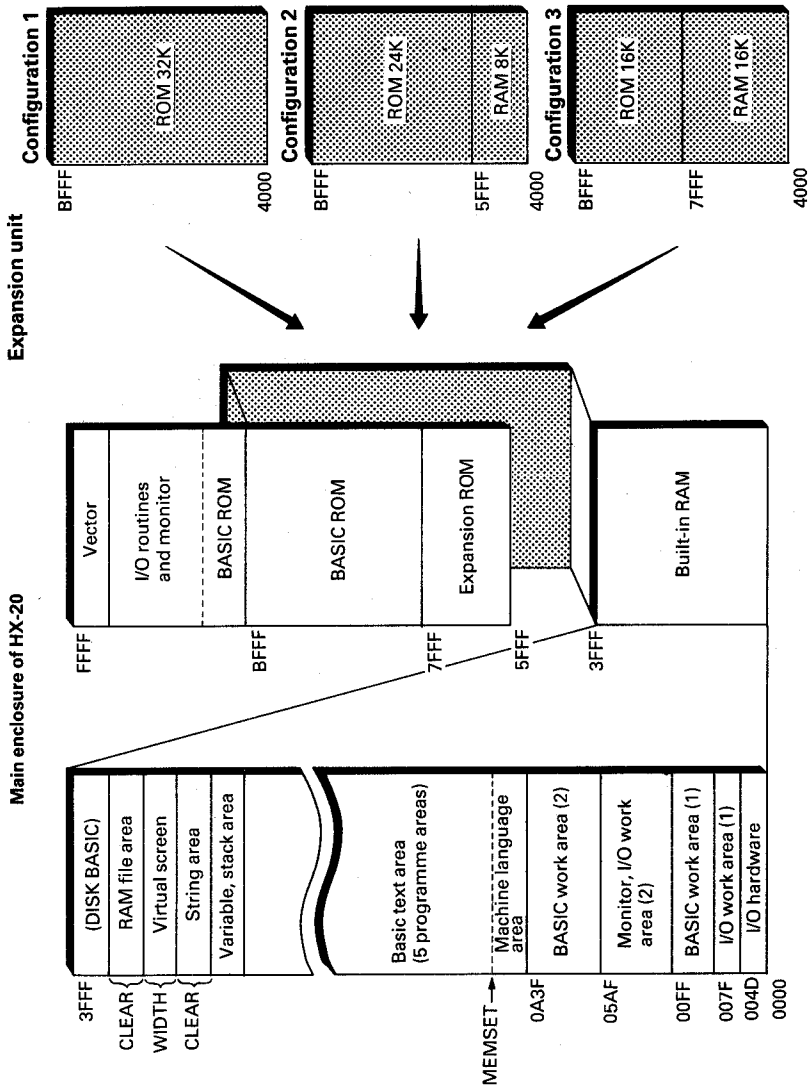
Hex. No.	Binary No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	@ 64	P 80	` 96	p 112	┼ 128	○ 144	160	176	192	208	224	240
1	0001	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♠ 145	161	177	193	209	225	241
2	0010	2	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	♥ 146	162	178	194	210	226	242
3	0011	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6	22	& 38	6 54	F 70	V 86	f 102	v 118	│ 134	ª 150	166	182	198	214	230	246
7	0111	7	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	↓ 151	167	183	199	215	231	247
8	1000	8	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	↑ 152	168	184	200	216	232	248
9	1001	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	Y 153	169	185	201	217	233	249
A	1010	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	→ 154	170	186	202	218	234	250
B	1011	11	27	+ 43	; 59	K 75	[91	k 107	{ 123	▒ 139	← 155	171	187	203	219	235	251
C	1100	12	28	, 44	< 60	L 76	Ñ 92	l 108	ñ 124	■ 140	↓ 156	172	188	204	220	236	252
D	1101	13	29	- 45	= 61	M 77	¿ 93	m 109	} 125	▬ 141	× 157	173	189	205	221	237	253
E	1110	14	30	. 46	> 62	N 78	^ 94	n 110	~ 126	▌ 142	÷ 158	174	190	206	222	238	254
F	1111	15	31	/ 47	? 63	O 79	_ 95	o 111	127	● 143	± 159	175	191	207	223	239	255

APPENDIX G Control Codes

Decimal	Hexadecimal	Function	Keys
1	01	Moves the physical screen to the left corner of the virtual screen.	CTRL + A
3	03	Escape AUTO mode.	CTRL + C
4	04	Moves the physical screen to the right.	CTRL + D, CTRL + →
5	05	Deletes all characters to the end of the line.	CTRL + E
6	06	Moves the physical screen to the right corner of the virtual screen.	CTRL + F
8	08	Backspace	CTRL + H, INS DEL
9	09	TAB (spaces every 8 columns)	CTRL + I, TAB
10	0A	Line feed	CTRL + J
11	0B	Moves the cursor to its home position.	CTRL + K, SHIFT + HOME CLR
12	0C	Clears the virtual screen.	CTRL + L, HOME CLR
13	0D	Carriage return	CTRL + M, RETURN
16	10	Moves the physical screen up.	CTRL + P SC ⋮ RN
17	11	Moves the physical screen down.	CTRL + Q, SHIFT + SC ⋮ RN
18	12	Insert mode	CTRL + R, SHIFT + INS DEL
19	13	Moves the physical screen to the left.	CTRL + S, CTRL + ←
22	16	Turns the cursor ON.	CTRL + V
23	17	Turns the cursor OFF.	CTRL + W
26	1A	Deletes all characters from the cursor position to the bottom line of the virtual screen.	CTRL + Z
28	1C	Moves the cursor to the right.	→
29	1D	Moves the cursor to the left.	←
30	1E	Moves the cursor up.	SHIFT + ↑
31	1F	Moves the cursor down.	SHIFT + ↓

Character codes 0 to 31 are not displayed as characters even if they are in an output statement. However, if they are included in a string, each character code will be counted as a one-character length.

APPENDIX H Memory Map



APPENDIX I
Table of Reserved Words

ABS	ERL	LPRINT	SAVE
ALL	ERR	MEMSET	SAVEM
AND	ERROR	MERGE	SCREEN
ASC	EXEC	MID$	SCROLL
ATN	EXP	MOD	SGN
AUTO	FILES	MON	SIN
BASE	FIX	MOTOR	SOUND
CDBL	FN	NEW	SPACE$
CHR$	FOR	NEXT	SPC
CINT	FRE	NOT	SQR
CLEAR	GCLS	OCT$	STAT
CLOSE	GET	OFF	STEP
CLS	GO	ON	STOP
COLOR	HEX$	OPEN	STR$
CONT	IF	OPTION	STRING$
COPY	IMP	OR	SUB
COS	INKEY$	PCOPY	SWAP
CSNG	INPUT	PEEK	TAB
CSRLIN	INSTR	POINT	TAN
DATA	INT	POKE	TAPCNT
DATE	KEY	POS	THEN
DAY	LEFT$	PRESET	TIME
DEF	LEN	PRINT	TITLE
DEFDBL	LET	PSET	TO
DEFFIL	LINE	PUT	TROFF
DEFINT	LIST	RANDOMIZE	TRON
DEFSNG	LLIST	READ	USING
DEFSTR	LOAD	REM	USR
DELETE	LOAD?	RENUM	VAL
DIM	LOADM	RESTORE	VARPTR
ELSE	LOCATE	RESUME	WEND
END	LOCATES	RETURN	WHILE
EOF	LOF	RIGHT$	WIDTH
EQV	LOG	RND	WIND
ERASE	LOGIN	RUN	XOR

APPENDIX J
List of Commands and Statements

AUTO

FORMAT AUTO [<line number>][,[<increment>]]
PURPOSE To generate a line number automatically.
EXAMPLE AUTO 100, 10
AUTO 200,
AUTO 300
AUTO

CLEAR

FORMAT CLEAR [<character area size>[,<RAM file size>]]
PURPOSE To initialize variables and to set the size of the character area and the RAM file.
EXAMPLE CLEAR 200, 256

CLOSE

FORMAT CLOSE[[#]<file number>[,[#]<file number>...]]
PURPOSE To close file(s).
EXAMPLE CLOSE #3

*CLS

FORMAT CLS
PURPOSE To clear a text screen.
EXAMPLE CLS

COLOR

FORMAT COLOR [<foreground color>][,[<background color>] [,<color set>]]
PURPOSE To specify the screen colors of the external display.
EXAMPLE COLOR 0, 3, 0

CONT

FORMAT CONT
PURPOSE To resume the execution of a programme that has been stopped.
EXAMPLE CONT

COPY

FORMAT COPY
PURPOSE To output the characters and graphics displayed on the LCD, on the built-in microprinter.
EXAMPLE COPY

DATA

FORMAT DATA <constant>[,<constant>...]
PURPOSE To store the numeric and string constants that are accessed by the READ statement(s).
EXAMPLE DATA HX, 20, EPSON

DEFFIL

FORMAT DEFFIL <record length>, <relative address>
PURPOSE To define the relative address of record 0 in a RAM file and the length of a single record.
EXAMPLE DEFFIL 20, 200

DEF FN

FORMAT DEF FN<name>[(<parameter>[,<parameter>....])] =<function definition>
PURPOSE To define a function created by the user.
EXAMPLE DEF FNZ(X, Y)=X*2+Y*3+A

DEFINT/SNG/DBL/STR

FORMAT DEF | INT / SNG / DBL / STR | <range(s) of letters>
PURPOSE To declare variable types as integer, single precision, double precision, and string.
EXAMPLE DEFSTR A,X–Z

DEF USR

FORMAT DEF USR[<digit>]=<starting address>
PURPOSE To specify the starting address of a machine language subroutine.
EXAMPLE DEF USR6=&H0C00

DELETE

FORMAT DELETE [<starting line number>][– [<ending line number>]]
PURPOSE To delete specified programme lines.
EXAMPLE DELETE 100–200
DELETE 100–
DELETE –200
DELETE 100

DIM

FORMAT DIM<variable>(<maximum subscript value>[,<maximum subscript value>...])[,...]
PURPOSE To declare the size of array variable elements.
EXAMPLE DIM A(40, 10), B$(50)

END

FORMAT END
PURPOSE To close all files and terminate programme execution.
EXAMPLE END

ERASE

FORMAT ERASE <array variable>[,<array variable>...]
PURPOSE To eliminate arrays from a programme.
EXAMPLE ERASE A, B

ERROR

FORMAT ERROR<integer expression>
PURPOSE To simulate the occurrence of an error; or to allow error codes to be defined by the user.
EXAMPLE ERROR 225

EXEC

FORMAT EXEC [<starting address>]
PURPOSE To start execution of a machine language programme.
EXAMPLE EXEC &H0C00

FILES

FORMAT FILES["<device name>"]
PURPOSE To display the names of all files residing on a specified memory device.
EXAMPLE FILES "CAS1:"

FOR...TO...STEP-NEXT

FORMAT FOR <variable>=<initial value> TO <final value> [STEP <increment>]
.
.
.
NEXT [<variable>[,<variable>]]

PURPOSE To allow a series of instructions between FOR and NEXT statements to be performed in a loop a given number of times.

EXAMPLE FOR I=0 TO 100 STEP 5
.
.
.
NEXT I

GCLS

FORMAT GCLS

PURPOSE To clear a graphic screen.

EXAMPLE GCLS

GET%

FORMAT GET% <record number>,<variable name>[,<variable name>...]

PURPOSE To read data from a RAM file into variables.

EXAMPLE GET% 0, A!, B#, C$

GOSUB...RETURN

FORMAT GOSUB <line number>
.
.
.
RETURN

PURPOSE To branch to and return from a subroutine.

EXAMPLE GOSUB 500

GO TO/GOTO

FORMAT (1) GO TO <line number>, or
(2) GOTO <line number>

PURPOSE To branch programme execution to a specified line number.

EXAMPLE GOTO 300

IF...THEN...ELSE/IF...GOTO...ELSE

FORMAT IF <expression>
{THEN {<statement> / <line No.>} / GOTO <line No.>} [ELSE {<statement> / <line No.>}]

PURPOSE To choose a particular route for programme execution based on conditions established in an expression.

EXAMPLE IF A >10 THEN A=0 ELSE 200

INPUT

FORMAT INPUT ["<prompt string>"{; / ,}]<variable>
[,<variable>...]

PURPOSE To allow input from the keyboard into a specified variable during programme execution.

EXAMPLE INPUT"NAME"; A$

INPUT#

FORMAT INPUT# <file number>, <variable>[,<variable ...]

PURPOSE To read data items from a sequential file and assign them to programme variables.

EXAMPLE INPUT#1, A, B, C$

KEY

FORMAT KEY <key number>, <string>

PURPOSE To define the programmable function keys.

EXAMPLE KEY 1,"LIST"

KEY LIST/KEY LLIST

FORMAT (1) KEY LIST
(2) KEY LLIST

PURPOSE To output the strings assigned to the programmable function keys on the screen and the microprinter, respectively.

EXAMPLE KEY LLIST

LET

FORMAT [LET]<variable>=<expression>

PURPOSE To assign the value of an expression to a variable.

EXAMPLE LET A=3.141592

LINE

FORMAT LINE[(<horizontal coordinate 1>, <vertical coordinate 1>)]–(<horizontal coordinate 2>, <vertical coordinate 2>), {PSET / PRESET} [,<colour>]

PURPOSE To draw a straight line between two specified points.

EXAMPLE LINE(0,0)–(119,31),PSET

LINE INPUT

FORMAT LINE INPUT ["<prompt string>";]<string variable>

PURPOSE To input an entire line to a string variable.

EXAMPLE LINE INPUT "WHAT?";A$

LINE INPUT#

FORMAT LINE INPUT#<file number>,<string variable>

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

EXAMPLE LINE INPUT #1, A$

LIST/LLIST

FORMAT (1) LIST[<starting line number>][–[<ending line number>]]
(2) LLIST[<starting line number>][–[<ending line number>]]

PURPOSE To output a programme list (1) on the LCD or external display or (2) on the microprinter.

EXAMPLE LIST 100–200
LIST –200
LIST 100–
LIST 200
LIST
LIST

LIST <file descriptor>

FORMAT LIST<file descriptor>[,[<line number>] [–[<line number>]]]

PURPOSE To output a programme list into a specified file.

EXAMPLE LIST "COM0:"

*LIST "COM0:"

FORMAT LIST"COM0:[(<BLPSC>)]"[, [<line number>][–[<line number>]]]

PURPOSE To specify the interface conditions of the RS-232C port and execute LIST.

EXAMPLE LIST"COM0:(2701B)"

LOAD

FORMAT LOAD[<file descriptor>[,R]]
PURPOSE To load a programme file into the memory.
EXAMPLE LOAD"CAS1:PROG1.ASC"

*LOAD "COM0:"

FORMAT LOAD"COM0:[(<BLPSC>)]"
PURPOSE To specify the interface conditions of the RS-232C port and execute LOAD.
EXAMPLE LOAD"COM0:(68N2B)"[,R]

LOADM

FORMAT LOADM<file descriptor>][,[offset value][,R]]
PURPOSE To load machine language programme file into the memory.
EXAMPLE LOADM"CAS1:ABC"

*LOAD?

FORMAT LOAD?[<file descriptor>]
PURPOSE To check files.
EXAMPLE LOAD?"CAS1:PROG1.ASC"

*LOCATE

FORMAT LOCATE<horizontal coordinate>,<vertical coordinate>[,<cursor switch>]
PURPOSE To specify the cursor position on the screen.
EXAMPLE LOCATE 10, 10, 0

*LOCATES

FORMAT LOCATES <horizontal coordinate>, <vertical coordinate>
PURPOSE To specify the position of the physical screen.
EXAMPLE LOCATES 0, 0

*LOGIN

FORMAT LOGIN <expression>[,R]
PURPOSE To switch the programme areas.
EXAMPLE LOGIN 3

*MEMSET

FORMAT MEMSET [<bottom address of memory>]
PURPOSE To specify the lower limit of the memory.
EXAMPLE MEMSET &HOD00

MERGE

FORMAT MERGE [<file descriptor>[,R]]
PURPOSE To merge a specified programme file into the programme currently in memory.
EXAMPLE "CAS1:PROG3.ASC"

*MERGE "COM0:"

FORMAT MERGE "COM0:[(<BLPSC>)"[,R]]
PURPOSE To specify the interface conditions of the RS-232C port and to execute MERGE.
EXAMPLE MERGE"COM0:(68N2B)",R

MID$

FORMAT MID$ (<string exp 1>,<n>[,<m>])=<string exp 2> where n and m are integer expressions and <string exp 1> and <string exp 2> are string expressions.
PURPOSE To replace a portion of one string with another string.
EXAMPLE MID$(A$,2)="BASIC"

MON

FORMAT MON
PURPOSE To transfer programme control to the machine language monitor.
EXAMPLE MON

MOTOR

FORMAT MOTOR [<switch>]
PURPOSE To turn ON/OFF the motor of the external audio cassette.
EXAMPLE MOTOR ON

*NEW

FORMAT NEW
PURPOSE To delete the programme in the memory and clear all variables.
EXAMPLE NEW

ON ERROR GOTO

FORMAT ON ERROR GOTO <line number>
PURPOSE To enable error trapping and specify the first line of the error handling subroutine.
EXAMPLE ON ERROR GOTO 1000

ON...GOSUB/ON...GOTO

FORMAT ON <expression> | GOSUB / GOTO | <line number> [,<line number>...]
PURPOSE To branch to one of several specified line numbers.
EXAMPLE ON A GOSUB 100, 200, 300, 400
ON B GOTO 100, 200, 300, 400

OPEN

FORMAT OPEN"<mode>", [#] <file number>, <file descriptor>
PURPOSE To open a specified file for I/O.
EXAMPLE OPEN"O", #1 "CAS0:TEST.BAS"

*OPEN"COM0:"

FORMAT OPEN "<mode>", [#]<file number>, "COM0:[(<BLPSC>)]"
PURPOSE To specify the interface conditions for the RS-232C port and execute OPEN.
EXAMPLE OPEN "O", #1,"COM0:(68N2B)"

OPTION BASE

FORMAT OPTION BASE | 0 / 1 |
PURPOSE To declare the minimum value for array variable subscripts.
EXAMPLE OPTION BASE 1

*PCOPY

FORMAT PCOPY <expression>
PURPOSE To copy a BASIC programme into another programme area.
EXAMPLE PCOPY 3

*POKE

FORMAT POKE <address>, <numeric expression>
PURPOSE To write a byte into a specified memory location.
EXAMPLE POKE &HOCOO, &H39

PRESET

FORMAT PRESET (<horizontal coordinate>, <vertical coordinate>)
PURPOSE To erase a dot on a graphic screen.
EXAMPLE PRESET (40,25)

PRINT/LPRINT

FORMAT PRINT / LPRINT [<expression>[;/,|<expression>...]]

PURPOSE To output data on the screen or the built-in microprinter.

EXAMPLE PRINT "EPSON"

PRINT USING/LPRINT USING

FORMAT PRINT / LPRINT USING <format string>;
[<expression>[;/,|<expression> ...]]

PURPOSE To output strings or numerics using a specified format.

EXAMPLE PRINT USING "####"; A,B

*PRINT#

FORMAT PRINT# <file number>, [<expression>...]

PURPOSE To write data into a sequential file.

EXAMPLE PRINT#1,A,B

PRINT# USING

FORMAT PRINT #<file number>, USING<format string>;
[<expression>[;/,|<expression>...]]

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

EXAMPLE PRINT#1, USING"###"; A

PSET

FORMAT PSET (<horizontal coordinate>, <vertical coordinate>)[,<colour>]

PURPOSE To draw dots on a specified graphic screen.

EXAMPLE PSET (30,20)

*PUT%

FORMAT PUT% <record number>, <variable>[,<variable>...]

PURPOSE To write the values of variables into a RAM file.

EXAMPLE PUT%0, A!, B#, C$

RANDOMIZE

FORMAT RANDOMIZE [<expression>]

PURPOSE To reseed the random number generator.

EXAMPLE RANDOMIZE

READ

FORMAT READ<variable>[,<variable>...]

PURPOSE To read values from a DATA statement and assigning them to variables.

EXAMPLE READ A, I, C$

REM

FORMAT REM[<remark>]

PURPOSE To allow explanatory remarks to be inserted in a programme.

EXAMPLE REM COMMENT MESSAGE

RENUM

FORMAT RENUM [<new number>][,<old number>][,<increment>]]

PURPOSE To renumber programme lines.

EXAMPLE RENUM

RESTORE

FORMAT RESTORE [<line number>]

PURPOSE To allow DATA statements to be reread from a specified point.

EXAMPLE RESTORE 1000

RESUME

FORMATS RESUME [/ NEXT / <line number>]

PURPOSE To continue programme execution after an error recovery procedure has been performed.

EXAMPLE RESUME 100

RUN

FORMAT (1) RUN [<line number>], or
(2) RUN <file descriptor>[,R]

PURPOSE To start programme execution.

EXAMPLE (1) RUN 300
(2) RUN"CAS0:PROG4.ASC"

RUN"COM0:"

FORMAT RUN"COM0:[(<BLPSC>)]"[,R]

PURPOSE To specify the interface condition of the RS-232C port and execute RUN.

EXAMPLE RUN"COM0:(68N2B)"

SAVE

FORMAT SAVE <file descriptor>[,A][,V]

PURPOSE To save an EPSON BASIC programme on a specified file.

EXAMPLE SAVE"CAS0:ABC"

*SAVE"COM0:"

FORMAT SAVE"COM0:[(<BLPSC>)]", A

PURPOSE To specify the interface conditions of the RS-232C port and execute SAVE.

EXAMPLE SAVE"COM0: (68E13)",A

*SAVEM

FORMAT SAVEM <file descriptor>,<top address>,<bottom address> <execution starting address>[,V]

PURPOSE To save the memory contents on a specified file.

EXAMPLE SAVEM"CAS1:ABC", &H0B00, &H0C00, &H0B00

*SCREEN

FORMAT SCREEN <text>, <graphic mode>

PURPOSE To specify the text or graphic screen modes.

EXAMPLE SCREEN 0,2

*SCROLL

FORMAT SCROLL[<speed>][,<mode>][,<scroll step X>,<scroll step Y>]]

PURPOSE To specify the SCROLL function of the physical screen.

EXAMPLE SCROLL 9,0,10,4

*SOUND

FORMAT SOUND <tone>,<duration>

PURPOSE To sound a specified tone.

EXAMPLE SOUND 10, 10

*STAT

FORMAT STAT [/ ALL / <expression>]

PURPOSE To display the status of each programme area.

EXAMPLE STAT 3

STOP

FORMAT STOP

PURPOSE To terminate programme execution and return to command level.

EXAMPLE STOP

SWAP

FORMAT SWAP <variable 1>,<variable 2>
PURPOSE To exchange the values of two variables.
EXAMPLE SWAP A$, B$

*TITLE

FORMAT TITLE <programme name>
PURPOSE To name programmes.
EXAMPLE TITLE"TEST 1"

TRON/TROFF

FORMAT TRON
TROFF
PURPOSE To trace the execution of programme statements.
EXAMPLE TRON
TROFF

*WIDTH

FORMAT WIDTH <characters per line>, <number of lines> [,<scroll margin>]
PURPOSE To set the size of the virtual screen.
EXAMPLE WIDTH 20, 25, 5

WIDTH <device name>

FORMAT WIDTH { "LPT0:" / "COM0:" }, <number of digits>
PURPOSE To set the print width of the printer.
EXAMPLE WIDTH"LPT0:", 20

WIND

FORMAT WIND[<counter value>]
PURPOSE To control the microcassette drive for fast forward and rewind.
EXAMPLE WIND 0

Functions

ABS

FORMAT ABS(<numeric expression>)
PURPOSE To return the absolute value of a numeric expression
EXAMPLE A=ABS(−1.6)

ASC

FORMAT ASC(<string>)
PURPOSE To return the character code of a character.
EXAMPLE A=ASC("A")

ATN

FORMAT ATN(<numeric expression>)
PURPOSE To return the arc tangent of a numeric expression.
EXAMPLE A=ATN(0.5)

CDBL

FORMAT CDBL(<numeric expression>)
PURPOSE To convert integers and single precision numbers into double precision numbers.
EXAMPLE A#=CDBL(B!/2)

CHR$

FORMAT CHR$(<numeric expression>)
PURPOSE To return the character corresponding to a specified character code.
EXAMPLE A$=CHR$(&H41)

CINT

FORMAT CINT(<numeric expression>)
PURPOSE To convert single and double precision numbers into integers.
EXAMPLE A%=CINT(B#/2)

COS

FORMAT COS (<numeric expression>)
PURPOSE To return the cosine of a numeric expression.
EXAMPLE A=COS(3.1415926/2)

CSNG

FORMAT CSNG(<numeric expression>)
PURPOSE To convert integers and double precision numbers into single precision numbers.
EXAMPLE A!=CSNG(B#)

CSRLIN

FORMAT CSRLIN
PURPOSE To return the vertical position of the cursor on the virtual screen.
EXAMPLE Y=CSRLIN

DATE$

FORMAT DATE$ [=MM/DD/YY]
PURPOSE To set the current date in, and return the date kept by, the internal calendar clock.
EXAMPLE PRINT DATE$

DAY

FORMAT DAY
PURPOSE To set the current day of the week in, and display the day of the week kept by, the internal calendar clock.
EXAMPLE PRINT DAY

EOF

FORMAT EOF(<file number>)
PURPOSE To return the end-of-file code.
EXAMPLE IF EOF(3) THEN CLOSE #1 ELSE GOTO 100

ERL/ERR

FORMAT ERL
ERR
PURPOSE To return the error code of an occurred error and the line number where the error occurred.
EXAMPLE A=ERL
B=ERR

EXP

FORMAT EXP (<numeric expression>)
PURPOSE To return the value of an exponential with e as its base.
EXAMPLE A=EXP(1)

FIX

FORMAT FIX (<numeric expression>)
PURPOSE To return the truncated integer part of a numeric expression.
EXAMPLE A=FIX(−B/3)

FRE

FORMAT FRE(<expression>)
PURPOSE To return the size of an unused memory area.
EXAMPLE PRINT FRE(0)
PRINT FRE("A$")

HEX$

FORMAT HEX$(<numeric expression>)
PURPOSE To return a string which represents the hexadecimal value of the decimal argument.
EXAMPLE A$=HEX$(65535)

INKEY$

FORMAT INKEY$
PURPOSE To return a one-character string of the pressed character key or a null string if no character key is pressed.
EXAMPLE A$=INKEY$

INPUT$

FORMAT INPUT$(<number of characters>[,[#]<file number>])
PURPOSE To return a string of characters read from a specified file.
EXAMPLE A$=INPUT$(5,#3)

INSTR

FORMAT INSTR([<numeric expression>,]<string 1>, <string 2>)
PURPOSE To search for the first occurrence of one string in another string and returns the position of the searched string.
EXAMPLE B=INSTR(A$,"XYZ")

INT

FORMAT INT(<numeric expression>)
PURPOSE To return the largest integer value (truncated).
EXAMPLE PRINT INT(-B/3)

LEFT$

FORMAT LEFT$(<string>,<numeric expression>)
PURPOSE To return an arbitrary length of string from the leftmost characters of a string.
EXAMPLE B$=LEFT$(A$,4)

LEN

FORMAT LEN(<string>)
PURPOSE To return the total number of characters in a string.
EXAMPLE A=LEN(A$)

LOF

FORMAT LOF (<file number>)
PURPOSE To return the size of a specified file.
EXAMPLE A=LOF(3)

LOG

FORMAT LOG(<numeric expression>)
PURPOSE To return the natural logarithm of a numeric expression.
EXAMPLE PRINT LOG(2.7812818)

MID$

FORMAT MID$(<string>,<expression 1>[,<expression 2>])
PURPOSE To return an arbitrary length of string from a string.
EXAMPLE B$=MID$(A$,2,3)

OCT$

FORMAT OCT$(<numeric expression>)
PURPOSE To return a string which represents the octal value of the decimal argument.
EXAMPLE PRINT OCT$(123+456)

*PEEK

FORMAT PEEK(<address>)
PURPOSE To return the byte read from a specified memory location.
EXAMPLE A=PEEK(&H0C00)

POINT

FORMAT POINT(<horizontal coordinate>,<vertical coordinate>)
PURPOSE To return the status of a dot at a specified location on the graphic screen.
EXAMPLE PRINT POINT(100,10)

POS

FORMAT POS(<digit>)
PURPOSE To return the horizontal position of the cursor on the virtual screen or the horizontal position of the printer head.
EXAMPLE X=POS(0)

RIGHT$

FORMAT RIGHT$(<string>,<numeric expression>)
PURPOSE To return an arbitrary length of string from the rightmost characters of a string.
EXAMPLE PRINT RIGHT$("ABCD",3)

RND

FORMAT RND[(<numeric expression>)]
PURPOSE To return a random number.
EXAMPLE A=RND(1)

SGN

FORMAT SGN(<numeric expression>)
PURPOSE To return the sign of the value of a numeric expression.
EXAMPLE B=SGN(A)

SIN

FORMAT SIN(<numeric expression>)
PURPOSE To return the sine of a numeric expression.
EXAMPLE PRINT SIN(3.1415926/2)

SPACE$

FORMAT SPACE$(<numeric expression>)
PURPOSE To return a string of spaces of a specified length.
EXAMPLE A$="A"+SPACE$(10)+"C"

SPC

FORMAT SPC(<digit>)
PURPOSE To output a specified number of blanks.
EXAMPLE PRINT SPC(10);"A"

SQR

FORMAT SQR(<numeric expression>)
PURPOSE To return the square root of a numeric expression.
EXAMPLE A=SQR(2)

STR$

FORMAT STR$(<numeric expression>)
PURPOSE To return a string representation of the value of a numeric expression.
EXAMPLE A$=STR$(123)

STRING$

FORMAT STRING$ (<integer expression>, | <string expression> | <numeric expression> |)
PURPOSE To return a string of specified characters.
EXAMPLE PRINT STRING$(10,65)

*TAB

FORMAT TAB(<numeric expression>)
PURPOSE To space to a specified position on the line where the cursor is currently positioned.
EXAMPLE PRINT TAB(10);"ABC"

TAN

FORMAT TAN(<numeric expression>)
PURPOSE To return the tangent of a numeric expression.
EXAMPLE A=TAN(3.1416/4)

*TAPCNT

FORMAT TAPCNT
PURPOSE To return the value of the microcassette drive counter.
EXAMPLE PRINT TAPCNT
A=TAPCNT

TIME$

FORMAT TIME$="HH:MM:SS"
PURPOSE To return the time kept by the internal calendar clock.
EXAMPLE PRINT TIME$

USR

FORMAT USR[<digit>](<argument>)
PURPOSE To call a machine language subroutine defined by DEFUSR statement.
EXAMPLE A=USR 1(B)

VAL

FORMAT VAL(<string expression>)
PURPOSE To return the numerical value of a string expression.
EXAMPLE A=VAL("−123")

VARPTR

FORMAT VARPTR(<variable name>)
PURPOSE To return the address of a variable or array.
EXAMPLE PRINT HEX$(VARPTR(A))

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,UK
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