

**TECHNICAL REFERENCE MANUAL**

**PC-E500**

**SHARP CORPORATION 1990.3**  
INFORMATION SYSTEMS GROUP  
PERSONAL EQUIPMENT DIVISION

## Contents

<b>CHAPTER 1 FILE CONTROL SYSTEM .....</b>	<b>1</b>
How to call .....	1
Function .....	3
File Handle Table .....	4
Contents of error .....	5
Explanation of Each Function .....	6
Creating a file (00 <sub>H</sub> ) .....	6
Opening a file (01 <sub>H</sub> ) .....	6
Closing a file (02 <sub>H</sub> ) .....	7
Reading a block of the file (03 <sub>H</sub> ) .....	7
Writing a block of the file (04 <sub>H</sub> ) .....	8
Reading a byte of the file (05 <sub>H</sub> ) .....	8
Writing a byte of the file (06 <sub>H</sub> ) .....	9
Verifying a file (07 <sub>H</sub> ) .....	9
Nondestructive reading a file (08 <sub>H</sub> ) .....	10
Moving a file pointer (09 <sub>H</sub> ) .....	10
Reading various information of a file (0A <sub>H</sub> ) .....	11
Changing directory information of drive (0B <sub>H</sub> ) .....	11
Searching for corresponding file name (0C <sub>H</sub> ) .....	12
Renaming a file (0D <sub>H</sub> ) .....	12
Deleting a file (0E <sub>H</sub> ) .....	13
Reading a free capacity of drive (0F <sub>H</sub> ) .....	13
Initializing a file control system (10 <sub>H</sub> ) .....	13
File name character string .....	14
<b>CHAPTER 2 OUTLINE OF IOCS .....</b>	<b>15</b>
IOCS Entry .....	17
Device number .....	17
Command number .....	17
Error code .....	18
Device Driver Control Command (00 <sub>H</sub> -05 <sub>H</sub> ) .....	20
Checking the header address 1 (01 <sub>H</sub> ) .....	20
Checking the header address 2 (02 <sub>H</sub> ) .....	20
Checking the drive name (03 <sub>H</sub> ) .....	21
Processing the format (04 <sub>H</sub> ) .....	21
All initialize (05 <sub>H</sub> ) .....	21
Processing the File of Standard Character Device (08 <sub>H</sub> -0E <sub>H</sub> ) .....	22
Character device OPEN (08 <sub>H</sub> ) .....	22
Character device CLOSE (09 <sub>H</sub> ) .....	22

Read byte data (0C <sub>H</sub> ) .....	22
Write byte data (0D <sub>H</sub> ) .....	23
Read block data (0A <sub>H</sub> ) .....	23
Write block data (0B <sub>H</sub> ) .....	23
Nondestructive read byte data (0E <sub>H</sub> ) .....	24
File Processing of Standard Block Device (10 <sub>H</sub> -17 <sub>H</sub> ) .....	24
Media check (10 <sub>H</sub> ) .....	24
Media parameter block address (11 <sub>H</sub> ) .....	25
Read sector (12 <sub>H</sub> ) .....	26
Write sector (13 <sub>H</sub> ) .....	26
Write & verify sector (14 <sub>H</sub> ) .....	27
Verify sector (15 <sub>H</sub> ) .....	27
Status read (16 <sub>H</sub> ) .....	28
Get sector address (17 <sub>H</sub> ) .....	28
Processing of Special File Device (20 <sub>H</sub> -2F <sub>H</sub> ) .....	29
Other Devices (3F <sub>H</sub> -7F <sub>H</sub> ) .....	30
Format processing (3F <sub>H</sub> ) .....	30
IOCS initialization (40 <sub>H</sub> ) .....	30
Function peculiar to device (41 <sub>H</sub> -7F <sub>H</sub> ) .....	30
<b>CHAPTER 3 HOW TO USE EACH DEVICE</b> .....	<b>31</b>
No. 00 Icd Driver STDO: SCRN: .....	31
Format (3F <sub>H</sub> ) peculiar to E500 .....	32
Initializing LCD driver (40 <sub>H</sub> ) .....	33
One character output to arbitrary position (41 <sub>H</sub> ) .....	33
Setting cursor position (44 <sub>H</sub> ) .....	33
Character output to arbitrary position (42 <sub>H</sub> ) .....	34
Setting the type of cursor display (45 <sub>H</sub> ) .....	34
Symbol display (46 <sub>H</sub> ) .....	35
n line scroll-up (47 <sub>H</sub> ) .....	35
n line scroll-down (48 <sub>H</sub> ) .....	35
Clear line (49 <sub>H</sub> ) .....	36
8-dot pattern display (4A <sub>H</sub> ) .....	36
8-dot pattern read (4B <sub>H</sub> ) .....	36
1-dot display (4C <sub>H</sub> ) .....	37
1-dot read (4D <sub>H</sub> ) .....	37
Line display (4E <sub>H</sub> ) .....	37
Paint out of box (4F <sub>H</sub> ) .....	38
Setting of display state (50 <sub>H</sub> ) .....	38
Clearing of display 1 (51 <sub>H</sub> ) .....	39
Clearing of display 2 (52 <sub>H</sub> ) .....	39
n line insertion (54 <sub>H</sub> ) .....	39
Reading of one-line dot pattern (55 <sub>H</sub> ) .....	40
Writing of one-line dot pattern (56 <sub>H</sub> ) .....	40

Displaying regardless of range (57H) .....	40
Display, with guide line (58H) .....	41
Parameter work .....	41
No. 01 Key Driver (STDI: KYBD:) .....	43
Format (3FH) peculiar to E500 .....	43
Initialization of key device driver (40H) .....	44
Read out of matrix code (41H) .....	44
Non-destructive reading of matrix code (42H) .....	45
Key reading (43H) .....	46
Data set to keyboard buffer (44H) .....	47
Key clear (45H) .....	47
Display of function key (46H) .....	48
Parameter for key input device driver .....	48
No. 02 SIO (RS232C) Driver (COM:) .....	50
Format (3FH) .....	50
Initialization of each parameter (40H) .....	51
1-byte direct output (41H) .....	51
1-byte direct input (42H) .....	51
Setting of hardware (43H) .....	52
Setting of RS, RR, and ER ports (44H-49H) .....	52
Reading of CS and CD ports (4AH-4BH) .....	52
Parameter work .....	53
No. 03 Printer Driver (STDL:PRN:) .....	60
Format (3FH) .....	60
Initialization of each parameter (40H) .....	61
Printing data output (41H) .....	61
Read of printing position (42H) .....	61
Printer check (43H) .....	61
No. 04 Tape Driver (CAS:) .....	62
Write, read, and verify of the header block (44H-45H) .....	62
Initialization of each parameter (40H) .....	63
Write, read, and verify of the data block (41H-43H) .....	63
Format (3FH) .....	63
Parameter work .....	64
Structure of file .....	65
No. 05 RAM Disk Driver (E: F: G:) .....	67
Format (3FH) .....	67
Initialization of each parameter (40H) .....	68
No. 06 Memory Block Driver (S1: S2: S3) .....	69
Initialization of memory driver (40H) .....	69
Format (3FH) .....	70
Search for physical address (41H) .....	70
Condense (47H) .....	70
Change of block size (42H) .....	71

Rename of block (44H) .....	71
Transfer of block (43H) .....	72
Creation of memory block (45H) .....	72
Deletion of memory block (46H) .....	73
Creation of memory block 2 (48H) peculiar to E500 .....	73
Parameter of memory block device .....	74
Structure of slot .....	74
No. 07 2.5" FDD Driver (X: Y:) .....	75
Initialization of each parameter (40H) .....	75
No. 08 System Control Driver .....	76
Power off (41H) .....	76
Secure the work (42H) .....	77
Execution of bASIC (43H) .....	78
Acquire the address for processing from the intermediate code (45H, 46H) .....	78
No. 09 Function Driver .....	79
Type of Variable .....	83
Internal format of numeric value (at execution of operation) .....	83
Internal expression of character string .....	84
IOCS Special Technique .....	85
BASIC .....	87
Internal expression of BASIC program .....	87
Location to store data .....	90
Interrupt .....	91
List of interrupt factor .....	91

# CHAPTER 1 FILE CONTROL SYSTEM

---

File control system is the software to execute the file processing or input/output to/from the device. In BASIC programming, in addition to the ordinary file processing, input from key and output of PRINT statement are executed through this file transaction. Filing is performed using the file handle. Also, file and device are operated in the same way. Accordingly redirection of input/output is possible between the device and the file.

## How to call

To use the file control system, enter function number in i register and execute far call of 0FFE4H. For some functions, a, (cl), x, or y register may be used. "CALLF"

Input and output include characters, drawing line in the display and calculation of function.

This means every program has input and output. However, various levels (degree or stage) of input/output are required by the program.

Input/output of our pocket computer can be largely classified into three levels. Features of each level are as follows.

- Level 2: Level on which each device can be handled as a file. On this level, you can only open a file, receive the data and close the file.
- Level 1: Function of each device driver is utilized to its utmost on this level. Various functions can be used in accordance with device. You can handle a function etc. only on this level.
- Level 0: Hardware is operated directly on this level. The operation on this level is troublesome, but processing is very quick.

### Concrete example of level as to LCD device

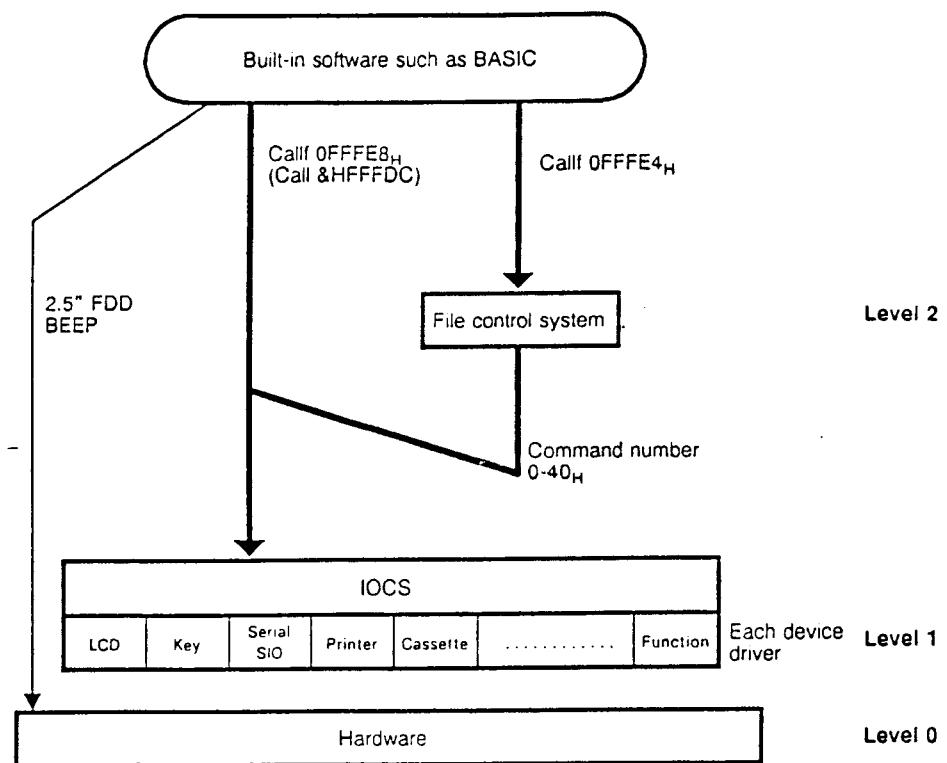
Level 2: It is possible to execute all processing that can be executed by OPEN statement in BASIC.

Level 1: Commands such as writing the character, drawing the line from the arbitrary position in the display or scrolling down etc. are supported to perform various processing.

Level 0: Every processing can be performed on this level. Actually it is rather difficult to operate LCD, though it is very quick.

Level is considerably related to compatibility of program with different type device.

## Relation of input and output



When creating program with this pocket computer, it is recommended to use level "0" (direct operation of hardware) only when you have to perform quick processing to correspond to succeeding device. (Actually, it is better not to use level 0.)

Following is the outline of each chapter.

CHAPTER 1: File control system (level 2)  
How to use F.C.S.

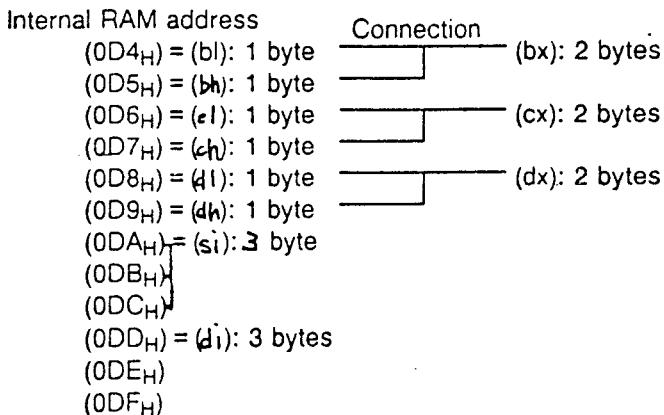
CHAPTER 2: Outline of IOCS (level 1)  
Instruction of structure/extension of device driver  
Explanation of call from file control system of device driver

CHAPTER 3: Use of each driver (level 1)  
Mainly, explanation how to use (command) of individual device driver.

*Supplement) —*

For transfer of parameter, symbols such as (cx), (dh) other than CPU register are used. These are the logic register existing in some specified position on the internal RAM, to make up the number of CPU registers.

1 byte: (bl), (bh), (cl), (ch), (dl), (dh)  
2 bytes: (bx), (cx), (dx)  
3 bytes: (si), (di)



## Function

The contents of each function are shown below.

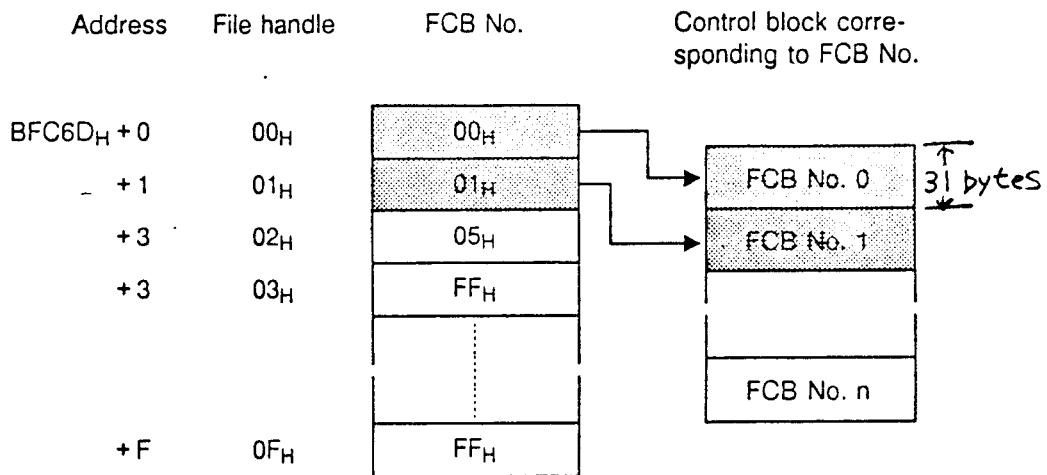
### Function No.

00H	Create a file
01H	Open a file
02H	Close a file
03H	Read a file block
04H	Write a file block
05H	Read a byte of file
06H	Write a byte of file
07H	Verify a file
08H	Nondestructive read of file
09H	Move a file pointer
0AH	Read of various information on file
0BH	Alteration of directory information of drive
0CH	Search of corresponding file name
0DH	Rename a file
0EH	Delete a file
0FH	Read of empty capacity of drive
10H	Initialization of file control system

## File Handle\_Table

File handle table is stored in the system memory and shows the relation of file handle and FCB number. File handle is the value returned from the file control system when a file is opened and it is a kind of #n of BASIC.

This value is used to read or write a command. The FCB is the table on which information for controlling file is written and indicates which is FCB number.



Here, file handles from 00H to 0FH and OFFH are included in the FCB number.

00H-0FH: FCB number  
FFH: unused

This file handle table can be altered directly with application. However, the following operations will cause incorrect execution of file control.

- (1) to write FCB number that is not open in the table.
- (2) to delete FCB number that is open from the table.

0, 1, and 2 of file handle are reserved by the system. When BASIC is activated, these file handles have been already opened and are assigned as follows.

Handle 0 = LCD display	(standard output)	std:
Handle 1 = key	(standard input)	stdi:
Handle 2 = printer	(standard listing output)	stdl:

---

## Contents of error

When an error occurred in each function, cf = 1 is obtained, it returns to a register with the following error codes.

- 00H An error occurred in the device and aborted.
- 01H The parameter is beyond the range.
- 02H The specified file does not exist.
- 03H The specified pass code does not exist.
- 04H The number of files to be opened exceeds the limit.
- 05H The file whose processing is not permitted.
- 06H Ineffective file handle was attempted.
- 07H Processing is not specified by open statement.
- 08H The file is during open.
- 09H The file name is duplicated.
- 0AH The specified drive does not exist.
- 0BH Error in data verification.
- 0CH Processing of byte number has not been completed.
- FEH Fatal low battery.
- FFH Processing has been interrupted. (break key was pressed.)

*cy = cf = carry flag*

## Explanation of Each Function

After entering function number and various parameter, execute fcall of FFFE4H. Explanations of each parameter are as follows.

### Creating a file (00H)

New file with the specified file name is created in the corresponding drive and is opened so that read and write are possible. When the file has been in the drive, open with the file size 0.

File attribute to be opened is given by putting value in a register. The file pointer is set at 000000H. Open as the read-out file in the read-out only device and as the write-in file in the write-in only device.

entry  
i = 00H  
a = file attribute  
    bit 0 write protect  
    bit 1 invisible  
x = the lead address of file name character string (00H in last with shift JIS)

return  
cf = 0  
    (cl) = File handle  
    a = File attribute  
cf = 1  
    Error (a = 00H, 03H, 04H, 05H, 08H, 0AH, FFH)

### Opening a file (01H)

Open a file of the specified file name. File pointer is set at 00000H. Even if the file is opened for writing-in, new file is not created. In case the file cannot be opened in the specified mode, an error occurs.

entry  
i = 01H  
a = 1 File is opened for reading-out.  
    2 File is opened for writing-in.  
    3 File is opened for reading and writing  
x = the lead address of file name character string (00H in last with shift JIS)

```

return
cf = 0
    (cl) = File handle
    a = File attribute
cf = 1
    Error (a = 00H, 01H, 02H, 03H, 04H, 05H, 08H, 0AH, FFH)

```

### Closing a file (02H)

Close a file of the specified file handle. Renewed directory information or FAT is written on the display. The file handle is released.

```

entry
    i = 02H
    (cl) = File handle

return
cf = 0
    No error
cf = 1
    Error (a = 00H, 06H, FFH)

```

### Reading a block of the file (03H)

Reading of datas with a specified number of bytes from files "in the specified file handle" and writing into specified memory.

The number of bytes can be chosen by using the code 1AH (end of file) which specifies the whole file or by specifying the number of bytes to be read.

```

entry
    i = 03H
    (cl) = File handle
    x = The lead address to which data is transferred.
    y = Number of bytes to be read
    a: bit 0 = 0 File end is 1AH code.
        (Pointer stops indicating 1AH. 1AH is read.)
        1 File end is a physical end of the file.
        (Pointer stops indicating final byte of pointer + 1)

return
cf = 0
    x = Next data of the read data
    y = Number of read bytes
cf = 1
    Error (a = 00H, 06H, 07H, 0CH, FFH)
    x = Next address that read correctly.
    y = Number of bytes that was read correctly.

```

### Writing a block of the file (04H)

Write the data of specified number of bytes in the file of the specified file handle from the specified memory.

Also, if the number of bytes to be written is set at 0 ( $y = 0$ ), the block from the file pointer to the file end is discarded.

entry

$i = 04H$   
 $(cl) = \text{File handle}$   
 $x = \text{The lead address of data}$   
 $y = \text{Number of bytes to be written}$

return

$cf = 0$   
 $x = \text{Next data of the written data}$   
 $y = \text{Number of written bytes}$   
 $cf = 1$   
 Error ( $a = 00H, 06H, 07H, 0CH, FFH$ )  
 $x = \text{Next address that was written correctly.}$   
 $y = \text{Number of bytes that was written correctly.}$

### Reading a byte of the file (05H)

Read 1-byte data in the register from the file of the specified file handle. You can specify whether the file end is set  $1AH$  code or in the physical file end.

entry

$i = 05H$   
 $(cl) = \text{File handle}$   
 $a:$  bit 0 = 0 File end is  $1AH$  code.  
 (Pointer stops indicating  $1AH$ .  $1AH$  is read.)  
 $a:$  bit 0 = 1 File end is a physical end of the file.  
 (Pointer stops indicating final byte of pointer + 1)

return

$cf = 0$   
 $a = \text{Data}$   
 $b = \text{Number of read data}$   
 $cf = 1$   
 Error ( $a = 00H, 06H, 07H, FFH$ )

**Writing a byte of the file (06H)**

Write 1-byte data to the file of the specified file handle.

```

entry
    i = 06H
    (cl) = File handle
    a = Data

return
    cf = 0
        b = Number of read data
    cf = 1
        Error (a = 00H, 06H, 07H, FFH)

```

**Verifying a file (07H)**

Read the data of number of specified bytes from the file of the specified file handle and verify the contents of memory.

You can specify whether the file end is set in 1AH code or in physical file end.

```

entry
    i = 07H
    (cl) = File handle
    x = The lead address of the data to be verified
    y = Number of bytes to be verified
    a: bit 0 = 0 File end is 1AH code.
        (Pointer stops indicating 1AH. 1AH is read.)
        1 File end is a physical end of the file.
        (Pointer stops indicating final byte of pointer + 1)

return
    cf = 0
        x = Next address of the data that was verified
        y = Number of bytes verified
    cf = 1
        Error (a = 00H, 06H, 07H, 0CH, FFH)
        x = Address of data that an error was occurred
        y = Number of bytes verified

```

## Nondestructive reading a file (08H)

Read 1-byte data into a register from the file of the specified file handle. File point does not move.

entry

i = 08H

(cl) = File handle

a: bit 0 = 0 File end is 1AH code.

(Pointer stops indicating 1AH. 1AH is read.)

1 File end is a physical end of the file.

(Pointer stops indicating final byte of pointer + 1)

bit 7 = 0 without data

1 with data

return

cf = 0

a = Data at the position on which the file pointer is.

b = Number of read bytes (0 or 1)

when number of bytes is 0, value of 'a' register is invalid.

cf = 1

Error (a = 00H, 06H, 07H, FFH)

## Moving a file pointer (09H)

Move the specified amount of the file pointer with the specified method. Value from 00000H to FFFFFH is specified.

In the write open mode, you can specify the value beyond file end. It is not possible to specify excessively in the read open mode.

entry

i = 09H

(cl) = File handle

(si) = Number of bytes to move (3 bytes)

a = 00H: Relative value from the file top (24 bits without sign)

01H: Relative value from present position (24 bits with sign)

02H: Relative value from file end + 1 (24 bits without sign)

return

cf = 0

(si) = File pointer value (3 bytes)

cf = 1

error (a = 00H, 01H, 06H, 07H, FFH)

## **Reading various information of a file (0AH)**

Read the file size, pointer value, file name, extension and attribute in the file that is open.

**entry**

i = 0AH  
(cl) = File handle  
a = 0: Reading of file size, pointer value  
1: Reading of file name, extension and attribute  
x = Address to read various information of file (when a = 1)

**return**

cf = 0  
when called out with (a = 0)  
a = Open attribute  
b = Device attribute  
(si) = File pointer value  
(di) = File size  
when called out with (a = 1)  
X + 0H = drive name + ":" (6 bytes)  
+ 6H = file name (8 bytes)  
+ EH = ". " (1 byte)  
+ FH = extension (3 bytes)  
+ 12H = attribute (1 byte)

cf = 1  
error (a = 00H, 06H, FFH)

## **Changing directory information of drive (0BH)**

Read and change the directory information of the drive. This is used for processing of files.

**entry**

i = 0BH  
a = 0 Reading of the directory information of drive  
1 Writing of the directory information of drive  
x = The lead address of a file name character string (00H in last  
~~with shift JIS~~ Japanese language only)  
y = The lead address of memory in which directory information is  
written (write directory information)

**return**

cf = 0  
y + 0 attribute byte  
+ 1 time information (all 00H when not specified)  
+ 3 date information (all 00H when not specified)  
+ 5 file size (read only)

cf = 1  
error (a = CCH, 01H, 02H, 03H, 08H, 0AH, FFH)

### Searching for corresponding file name (0C<sub>H</sub>)

Search for the file in the specified direction from the specified directory number. Wild card can be used for the file name character string to be searched for. Also, it is possible to specify disregard of error.

#### entry

i = 0C<sub>H</sub>

a = 00<sub>H</sub> Search for the back of the specified directory number

01<sub>H</sub> Search for the front of the specified directory number

80<sub>H</sub> Perform 00<sub>H</sub> disregarding to be invisible

81<sub>H</sub> Perform 01<sub>H</sub> disregarding to be invisible

(bx) = Directory number to start searching (This position is also searched for.)

x = Lead address of file name character string to be searched for. (00<sub>H</sub> in last with shift JIS)

y = Lead address to return the result

#### return

cf = 0

(bx) = Directory number of the file detected

x = Lead address of file name character string to be searched for

y = Lead address of file name character string that was detected

cf = 1

error (a = 00<sub>H</sub>, 03<sub>H</sub>, 09<sub>H</sub>, 0A<sub>H</sub>, FF<sub>H</sub>)

### Renaming a file (0D<sub>H</sub>)

Rename the file to the specified name.

#### entry

i = 0D<sub>H</sub>

x = Lead address of file name character string before renaming  
~~(00H in last with shift JIS)~~

y = Lead address of file name character string after renaming  
~~(00H in last with shift JIS)~~

#### return

cf = 0

none

cf = 1

error (a = 00<sub>H</sub>, 02<sub>H</sub>, 03<sub>H</sub>, 05<sub>H</sub>, 08<sub>H</sub>, 09<sub>H</sub>, 0A<sub>H</sub>, FF<sub>H</sub>)

## **Deleting a file (0E<sub>H</sub>)**

Delete the specified file.

entry  
    i = 0E<sub>H</sub>  
    x = Lead address of the file name character string to be deleted  
        (~~00H in last with shift JIS~~)

return  
    cf = 0  
        nil  
    cf = 1  
        error (a = 00<sub>H</sub>, 02<sub>H</sub>, 03<sub>H</sub>, 05<sub>H</sub>, 0A<sub>H</sub>, FF<sub>H</sub>)

## **Reading a free capacity of drive(0F<sub>H</sub>)**

Examine free capacity of the specified drive.

entry  
    i = 0F<sub>H</sub>  
    a = 0  
    x = Drive name character string (~~00H in last with shift JIS~~)

return  
    cf = 0  
        (si) = Free capacity of drive (3 bytes)  
    cf = 1  
        error (a = 00<sub>H</sub>, 03<sub>H</sub>, FF<sub>H</sub>)

## **Initializing a file control system (10<sub>H</sub>)**

Initialize the work, FCB, file handle table, file buffer in the file control system.

entry  
    i = 10<sub>H</sub>  
    a = 00<sub>H</sub>     Operation of initialization of the work and "a = 01<sub>H</sub>"  
              01<sub>H</sub>     Release all FCB, file handle, and file buffer. Open  
                         standard input, standard output, standard listing out  
                         put.  
              02<sub>H</sub>     Release file handle other than standard.

return  
    nil

### File name character string

File name character string consists of the drive name and file name

Drive name	:	File name	.	Extension 00H
1-5 bytes		8 bytes		3 bytes

Drive name is expressed in the byte from 1 to 5. End of drive name is marked with a colon.

File name is expressed in 8 bytes filling from the head. The space is used to fill vacancy. Put period (.) at end of the file name.

Extension is expressed in 3 bytes from the head. The space is used to fill vacancy. Vacancy may be filled with only space. Put 00H in last.

## CHAPTER 2 OUTLINE OF IOCS

---

IOCS is a software with which you can make efficient use of the hardware. In addition to a display and key, memory control and power control are provided as IOCS. As a compatibility will be given on this level in future, it is recommended to design your application software or device driver in accordance with this spec.

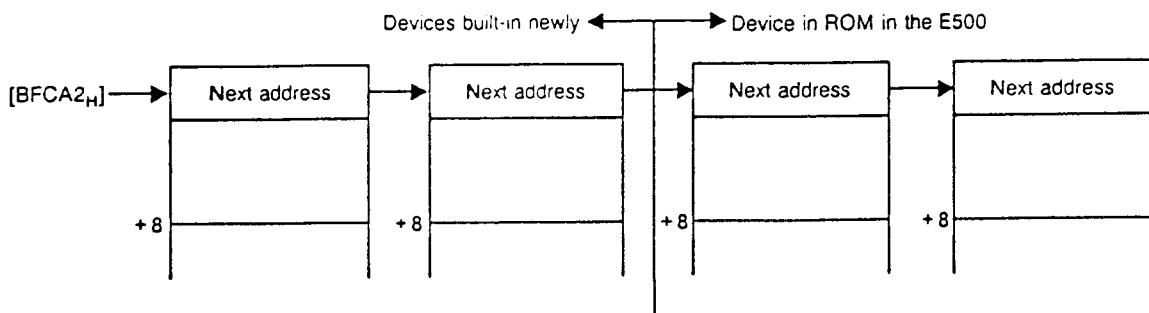
### CALUF

You can use IOCS by performing `feat#` of  $FFFE8H$ . Command is used to operate file-control and peculiar to device.

The IOCS consists of the IOCS main routine, IOCS routine of individual device, and IOCS header. As the IOCS is built in the system by the IOCS header, you can write a program assembled by the machine language and add the header to correspond with new device or to alter the function of existing device.

Structure of IOCS header		
+ 0	Address to next IOCS header	3 bytes
+ 3	Device number	1 byte
+ 4	Device attribute	1 byte
+ 5	Entry address of each IOCS	3 bytes
+ 8	Drive name n bytes	n byte
	List the drive name marked with a colon. 00H in last. Drive name is expressed in max. 5 bytes.	

- Address of next IOCS header  
Lead address of next IOCS header is shown. In the case of last header,  $FFFFFH$  is set.
- ADevice number  
The number to represent this device. Device number is peculiarly assigned to each device.



- Device attribute

The following information to show characteristics of device is stored.

Bit 7 = 1: device able to handle file-control

Bit 6 = 1: special file device (device that cannot be handled with standard processing by file-control)

Bit 5 = 0: block device (device controlled by cluster)

    1: character device

Bit 4 = 1: device that default is performed by ASCII code.

Bit 3 = 0

Bit 2 = 1: device that read and write cannot be executed simultaneously.

Bit 1 = 1: write enable device

Bit 0 = 1: read enable device

For example, C7H is the special device that enable read/write, but cannot execute read and write simultaneously.

- IOCS entry address

Address of IOCS entry. If BIOS call is executed, the desired header is detected from BIOS number and control is transferred to the address shown in BIOS main routine processing.

- Drive name

List the drive name (max. 5 bytes) enable to handle by the header marked off with a colon, and write 00H in the end. For example, if the drive name is "COM1:" and "COM2", indicate as follows.

43H, 4FH, 4DH, 31H, 3AH, 43H, 4FH, 4DH, 32H, 3AH, 00H

The drive number corresponds to the drive name. In the above example, "COM1" becomes 0, "COM2" to 1.

## IOCS Entry

### Device number

Device number is the peculiar number to represent the device. If the different numbers in every IOCS header and unable to handle are specified, an error occurs.

Device attribute is as follows.  $8 \times H$  is the standard block device,  $A \times H$  is standard character device and  $B \times H$ , and  $C \times H$  are special devices.

IOCS	Drive name	Device No.	Drive No.	Device attribute
Display	STD0:, SCRN:	0	0, 1	A2H
Key	STDI:, KYBD:	1	0, 1	A1H
SIO	COM:	2	0	D3H
Printer	STDL:, PRN:	3	0, 1	A2H
Tape	CAS:	4	0	D7H
Memory file	E:, F:, G:	5	0, 1, 2	83H
Memory card	S1:, S2:, S3:	6	0, 1, 2	C3H
2.5" FDD	X:	7	0	00H
System	SYSTM:	8	0	00H
Function	NIL	9	NIL	00H

can not available!

### Command number

IOCS entry command number consists of the part common to every device and the individual part of each device. Contents of the command are as follows.

- $00H - 07H$  Command for IOCS main routine. It does not jump to entry of each IOCS.
- $08H - 0FH$  File processing of standard character device. Use from file-control.
- $10H - 1FH$  File processing of standard block device. Use from file-control.
- $20H - 3FH$  File processing of special device. Use from file-control.
- $3FH$  Format processing
- $40H$  Initialization. Common to all devices
- $41H - 7FH$  Processing peculiar to each device
- $80H - FFH$  Reserve

### Error code

When an error occurred, each IOCS entry is returned with cy = 1. In this return, an error code is set in a register. Error codes are classified into 4 systems depending on the commands.

- Command No. = 00<sub>H</sub>-1F<sub>H</sub>, 40<sub>H</sub>

00<sub>H</sub> An attempt was made to write in the media of write protect.  
01<sub>H</sub> The drive does not exist.  
02<sub>H</sub> The drive is not ready.  
03<sub>H</sub> The command cannot be handled.  
04<sub>H</sub> The media has been changed.  
05<sub>H</sub> Write error  
06<sub>H</sub> Read error  
07<sub>H</sub> Verify error  
08<sub>H</sub> Device unable to write  
09<sub>H</sub> Device unable to read  
FE<sub>H</sub> Fatal low battery  
FF<sub>H</sub> Break was made.

- Command No. = 20<sub>H</sub>-3E<sub>H</sub>

00<sub>H</sub> An error occurred in the device and aborted.  
01<sub>H</sub> The parameter is beyond the range.  
02<sub>H</sub> The specified file does not exist.  
03<sub>H</sub> The specified pass code does not exist.  
04<sub>H</sub> The number of files to be opened exceeds the limit.  
05<sub>H</sub> The file processing is not allowed.  
06<sub>H</sub> Invalid file handle was used.  
07<sub>H</sub> Processing is not specified when opened.  
08<sub>H</sub> The file is during open.  
09<sub>H</sub> The corresponding file does not exist.  
0A<sub>H</sub> The file name is duplicated.  
0B<sub>H</sub> The specified drive does not exist.  
0C<sub>H</sub> Verify error  
0D<sub>H</sub> Processing of the specified number of bytes has not been completed.  
FE<sub>H</sub> Fatal low battery  
FF<sub>H</sub> Break was made.

- Command No. =  $3F_H$   
Same error code with BASIC
- Command No. =  $41_H$ - $7F_H$   
Refer to specifications of each device.

*cy = cf = carry flag*

### Device Driver Control Command (00H-05H)

#### Searching for the drive name (00H)

Search the device number and drive number from the specified drive name.

entry

i = 00H

x = drive name lead address

return

(cl) = device number

(ch) = drive number

cy = 0: no error

cy = 1: error (a = 01H)

#### Checking the header address 1 (01H)

Search the header address from the specified device number.

entry

i = 01H

(cl) = device number

return

cy = 0: x = header address

cy = 1: error (a = 01H)

#### Checking the Header Address 2 (02H)

Search the header address from the specified device number.

entry

i = 02H

(cl) = device number

x = the first header address to start searching for return

return

cy = 0: x = header address

cy = 1: error (a = 01H)

---

### Checking the drive name (03H)

Search the drive name from the specified device number and drive number.

entry

i = 03H  
(cl) = device number  
(ch) = drive number  
x = the lead address of area that returns drive name  
(6 bytes are required.)

return

cy = 0: drive name + colon is written from the position indicated by x, and filled with 20H.  
cy = 1: error (a = 01H)

### Processing the format (04H)

Execute format of the device of the specified drive name.

entry

i = 04H  
x = the lead address of (drive name + the set information character string)

return

cy = 0: x = next address processed as the set information character string  
cy = 1: error  
(a = 01H, same error code with BASIC)

### All initialize (05H)

Initialize all of the stored devices.

entry

i = 05H  
a = initialization level  
0: all reset  
1: reset  
2: off  
3: on

return

nil

---

## Processing the File of Standard Character Device (08H-0EH)

Common commands are provided for the character device that is able to handle file-control with common processing. By supporting the following commands, file-control can be handled as the standard character device.

### Character device OPEN (08H)

Execute initialization when the standard character device is opened.

```

entry
    i = 08H
    (cl) = device number
    (ch) = drive number
        x = the lead address of file name (12 bytes)
        a = OPEN mode
            1: for reading
            2: for writing
            3: for reading/writing
return
    cy = 1: error (a = 00H, 02H, 03H, 08H, 09H, FEH, FFH)

```

### Character device CLOSE (09H)

Execute processing when the standard character device is closed.

```

entry
    i = 09H
    (cl) = device number
    (ch) = drive number
return
    none

```

### Read byte data (0CH)

Read the data of 1 byte only.

```

entry
    i = 0CH
    (cl) = device number
    (ch) = drive number
return
    cy = 0: no error
        a = data
    cy = 1: error (a = 01H, 02H, 03H, 04H, 06H, 09H, FEH, FFH)

```

---

### Write byte data (0DH)

Write the data of 1 byte only.

```
entry
    i = 0DH
    (cl) = device number
    (ch) = drive number
    a = data
return
    cy = 0:  no error
    cy = 1:  error (a = 01H, 02H, 03H, 04H, 05H, 08H, FEH, FFH)
```

### Read block data (0AH)

Read out the data to the specified position by the specified number of bytes only.

```
entry
    i = 0AH
    (cl) = device number
    (ch) = drive number
    x = The lead address to which data is transferred.
    y = Number of bytes to be read
return
    x = Next address that read correctly.
    y = Number of bytes that was read correctly.
    cy = 0:  no error
    cy = 1:  error (a = 01H, 02H, 03H, 04H, 06H, 09H, FEH, FFH)
```

### Write block data (0BH)

Write out the data from the specified position by the specified number of byte only.

```
entry
    i = 0BH
    (cl) = device number
    (ch) = drive number
    x = The lead address of the data
    y = Number of bytes to be read
return
    x = Next address that read correctly.
    y = Number of bytes that was read correctly.
    cy = 0:  no error
    cy = 1:  error (a = 01H, 02H, 03H, 04H, 05H, 08H, FEH, FFH)
```

### Nondestructive read byte data (0EH)

Read data by only 1 byte. Data is returned but not deleted.

```
entry
    i = 0EH
    (cl) = device number
    (ch) = drive number
return
    cy = 0: no error
    a = data
    cy = 1: error (a = 01H, 02H, 03H, 04H, 06H, 09H, FEH, FFH)
```

### File Processing of Standard Block Device (10H-17H)

Common command is provided to block device in which file-control is handled with common processing. By supporting the following commands, file control can be used as the standard block device.

#### Media check (10H)

Store in memory the media that is currently installed for checking of the media change.  
Delete the contents stored.

```
entry
    i = 10H
    (cl) = device number
    (ch) = drive number
    a = 0: Media is stored in memory. After that, when the media is
          changed, an error occurs.
    1: Makes change of media possible.
    2: Check whether at present media has been changed or
       not.
return
    cy = 0: no error
    cy = 1: error (a = 01H, 02H, 03H, FEH, FFH)
```

## Media parameter block address (11H)

The lead address of several bytes which show the physical structure of media is returned.

```

entry
    i = 11H
    (cl) = device number
    (ch) = drive number
return
    cf = 0 : no error
    x = the lead address of media parameter block
    - (x + 0) = media descriptor byte (first byte of FAT)
    - (x + 1, 2) = number of bytes per sector (2 bytes of low and hi)
    - (x + 3) = (number of directories entered in a sector) - 1, but 32 bytes
      per directory.
    - (x + 4) = log2 (number of directories entered in a sector)
    - (x + 5) = 00H
    - (x + 6) = 01H
    - (x + 7, 8) = number of spare sectors (2 bytes of low and hi) - lead sector
      of FAT
    - (X + 9) = 08H
    - (x + A, B) = number of root directories (2 bytes of low and hi)
    - (x + C, D) = Lead sector of data (2 bytes of low and hi)
    - (x + E,F) = Number of usable sectors + 1 (2 bytes of low and hi)
    - (x + 10) = Number of sectors occupied with 1 FAT
    - (x + 11, 12) = Lead sector of directories (2 bytes of low and hi)
      However, number of bytes per sector = 2n × 32 (n = 1, 2, ...)
      number of directories contained in one sector = number of by-
      tes per sector/32
    Number of sectors occupied by one FAT = [number of usable sec-
      tors + 2]/1.5]
    Lead sector of directory = number of spare sectors + number of sectors
      occupied by one FAT
    Lead sector of data = lead sector of directory + [(number of root directo-
      ries/number of directories entered in one sector)]
    Number of usable sectors = total number of sectors - lead sector of data
    [x] = the smallest integer more than x
  
```

## Read sector(12H)

Only the number of specified sectors is read to the specified position.

entry  
    i = 12H  
    (cl) = device number  
    (ch) = drive number  
        x = address to which sector is transferred  
        (dx) = number of addresses to be transferred  
        (bx) = sector number to start transfer

return  
    cf = 0  
    x = address that transfer has been completed + 1  
    (dx) = number of sectors correctly transferred  
    (bx) = sector number that transfer has been completed + 1  
    cf = 1: error (a = 01H, 02H, 03H, 04H, 06H, 09H, FEH, FFH)  
        x = address of data in which an error occurred.  
    (dx) = number of sectors that has not been transferred correctly.  
    (bx) = sector number to continue transfer

## Write sector (13H)

Only number of bytes that was specified from the specified position is written.

entry  
    i = 13H  
    (cl) = device number  
    (ch) = drive number  
        x = address to which sector is transferred  
        (dx) = number of addresses to be transferred  
        (bx) = sector number to start transfer

return  
    cf = 0  
    x = address that transfer has been completed + 1  
    (dx) = number of sectors correctly transferred  
    (bx) = sector number that transfer has been completed + 1  
    cf = 1: error (a = 00H, 01H, 02H, 03H, 04H, 05H, 08H, FEH, FFH)  
        x = address of data in which an error occurred.  
    (dx) = number of sectors that has not been transferred correctly.  
    (bx) = sector number to continue transfer

## Write & verify sector (14H)

Only number of sectors that was specified from the specified position is written and after that is verified.

```
entry
    i = 14H
    (cl) = device number
    (ch) = drive number
        x = address to which sector is transferred
    (dx) = number of addresses to be transferred
    (bx) = sector number to start transfer

return
    cf = 0
    x = address that transfer has been completed + 1
    (bx) = sector number that transfer has been completed + 1
    cf = 1:   error (a = 00H, 01H, 02H, 03H, 04H, 05H, 07H, 08H, FEH,
               FFH)
    x = address of data in which an error occurred.
    (dx) = number of sectors that has not been transferred correctly.
    (bx) = sector number to continue transfer
```

## Verify sector (15H)

Only number of sectors that was specified from the specified position is verified.

```
entry
    i = 15H
    (cl) = device number
    (ch) = drive number
        x = address to which sector is transferred
    (dx) = number of sectors to be verified (number of bytes)
    (bx) = sector number to start to verify.

return
    cf = 0
    x = address that transfer has been completed + 1
    (bx) = sector number that transfer has been completed + 1
    cf = 1:   error (a = 01H, 02H, 03H, 04H, 07H, FEH, FFH)
    x = address of data in which an error occurred.
    (dx) = number of sectors that has not been transferred correctly.
    (bx) = sector number to continue transfer
```

**Status read (16H)**

The specified drive attribute is read.

```
entry
    i = 16H
    (cl) = device number
    (ch) = drive number
return
    cf = 0
    ba bit 0 = 0: non write protect drive
                = 1: write protect drive
    bit 1 = 0: ???
                = 1: ???
    cf = 1: error (a = 01H, 02H, 03H, FEH, FFH)
```

**Get sector address (17H)**

The address of the specified sector is got.

```
entry
    i = 17H
    (cl) = device number
    (ch) = drive number
    (bx) = sector number
return
    cf = 0
    x = lead address of the specified sector
    (cx) = sector size
    (di) = lead address of lead sector of the specified drive
    cf = 1: error (a = 01H, 02H, 03H, FEH, FFH)
```

## Processing of Special File Device (20<sub>H</sub>-2F<sub>H</sub>)

Execute processing of file control device that is neither standard character device nor standard block device.

Contents of entry operation and error code are same as file-control. For more details, refer the file-control.

- Command No. file-control

i = 20 <sub>H</sub>	00 <sub>H</sub> Creating a file
21 <sub>H</sub>	01 <sub>H</sub> Opening a file
22 <sub>H</sub>	02 <sub>H</sub> Closing a file
23 <sub>H</sub>	03 <sub>H</sub> Reading a block of the file
24 <sub>H</sub>	04 <sub>H</sub> Writing a block of the file
25 <sub>H</sub>	05 <sub>H</sub> Reading a byte of the file
26 <sub>H</sub>	06 <sub>H</sub> Writing a byte of the file
27 <sub>H</sub>	07 <sub>H</sub> Verifying a file
28 <sub>H</sub>	08 <sub>H</sub> Non-destructive reading a file
29 <sub>H</sub>	09 <sub>H</sub> Moving a file pointer
2A <sub>H</sub>	0A <sub>H</sub> Reading various information of the file
2B <sub>H</sub>	0B <sub>H</sub> Changing a drive directory information
2C <sub>H</sub>	0C <sub>H</sub> Searching for corresponding file name
2D <sub>H</sub>	0D <sub>H</sub> Renaming a file
2E <sub>H</sub>	0E <sub>H</sub> Deleting a file name
2F <sub>H</sub>	0F <sub>H</sub> Reading a drive space capacity

- Entry specification

Registers (cl), (ch), x, y, i, a... , same as file-control, are called after putting FCB No. in (dl). When using FCB as work, use this value as a reference.

## Other Devices ( $3F_H$ - $7F_H$ )

### Format processing ( $3F_H$ )

Format processing of device (media) common to all devices.

For example, card is initialized in RAM file, or parameter is set when COM: device. Format is called out with INIT in BASIC program.

```

entry
    i =  $3F_H$ 
    (cl) = device number
    (ch) = drive number
    x = lead address of set information character string
return
    cf = 0
    x = next address that was processed as set information character
        string.
    cf = 1: error (same error code as a = BASIC)

```

### IOCS initialization ( $40_H$ )

Each device is initialized in accordance with the specified level.

```

entry
    i =  $40_H$ 
    (cl) = device number
    (ch) = drive number
    a = 0: all reset    initialization of all parameters
          1: reset       initialization of some parameters
          2: off         initialization for "off"
          3: on          initialization for "on"
return
    cf = 0: no error
    cf = 1: error (a =  $01_H$ ,  $FE_H$ ,  $FF_H$ )

```

### Function peculiar to device ( $41_H$ - $7F_H$ )

Command for use of function peculiar to each device

## CHAPTER 3 HOW TO USE EACH DEVICE

This chapter describes how to use all device drivers equipped in the pocket computer as standard features.

The device driver supports two command groups in rough classification. One is the part called from the file control system explained in the previous chapter. And other is "operation particular to each device" detailed in this chapter.

After setting the specified register, etc., each device driver is called using IOCS explained in previous chapter.

And, command that requires only register il, (cl), (ch) is called after writing each value for address 0BFE00h<sub>H</sub> to 0BFE02h<sub>H</sub> with setting as follows.

CALL&HFFFD

For example, if called as follows, power is turned OFF.

POKE &HBFE00,8,0,&H41 :CALL&HFFFD

The following is command list and the method to use peculiar commands of each device driver separately shown.

The "NO." of the left end of device name indicates the device number particular to each device driver.

No. 00 LCD Driver STDO: SCRN:

cf = cy = carry flag

LCD driver is the following type of device.

- Device usable as a file.
- Standard character device
- Able to write only.
- STDO: LCD driver is opened as a standard output.

### Command list

- Command for standard character device
  - 08<sub>H</sub> No special processing Non-operation
  - 09<sub>H</sub> No special processing Non-operation
  - 0A<sub>H</sub> Error
  - 0B<sub>H</sub> Output to display
  - 0C<sub>H</sub> Error
  - 0D<sub>H</sub> Output to display
  - 0E<sub>H</sub> Error
  - 0F<sub>H</sub> Error

- Command for standard block device  
 $10H$ - $1FH$  Error
- Command for special device  
 $20H$ - $2FH$  Error
- Command peculiar to each device  
 $3FH$  Formatting  
 $40H$  Initialization of each parameter  
 $41H$  1 character output to arbitrary position  
 $42H$  n character output to arbitrary position  
 $43H$  Not used  
 $44H$  Sets cursor position  
 $45H$  Sets type of cursor display  
 $46H$  Performs symbol display  
 $47H$  n line scroll-up  
 $48H$  n line scroll-down  
 $49H$  Clears 1 line.  
 $4AH$  Displays 8-dot pattern.  
 $4BH$  Reads 8-dot patterns.  
 $4CH$  Displays 1 dot.  
 $4DH$  Reads 1 dot.  
 $4EH$  Displays line.  
 $4FH$  Paints out the box.  
 $50H$  Sets display state.  
 $51H$  Clears display.  
 $52H$  Clears from the specified position.  
 $53H$  (Deletes one line.)  
 $54H$  Inserts n line.  
 $55H$  Transfers one-line dot pattern to memory.  
 $56H$  Displays dot pattern on the memory.  
 $57H$  Displays regardless of display range.  
 $58H$  Displays guide line.

### Format ( $3FH$ )

entry  
(cx) =  $0000H$   
il =  $3FH$

return  
nil

## Initializing LCD driver (40H)

Initialize work area of LCD driver.

```
entry
  (cx) = 0000H
  i = 0040H
  a = level of initialization
    0: all reset
    1: reset
    2: off
    3: on

return
  nil
```

## One character output to arbitrary position (41H)

Output one character. After output, add 1 to x coordinate (bl). No processing is executed if it is outside range.

```
entry
  (cx) = 0000H
  i = 0041H
  (bl) = x coordinate at output position
  (bh) = y coordinate at output position
  a = output data

return
  (bl), (bh) = indicates next display position
  cy = 0: normal completion
  cy = 1: outside range
  a = output data
```

## Setting cursor position (44H)

Set the display position of cursor. Condition of cursor does not change.

```
entry
  (cx) = 0000H
  i = 0044H
  (bi) = cursor x coordinate
  (bh) = cursor y coordinate

return
  cy = An attempt was made to display outside range.
```

## Character output to arbitrary position(42H)

Character string is displayed. (bl) is displayed with adding one address. When reached right end of display, display is stopped.

entry

(cx) = 0000H  
i = 0042H  
(bl) = x coordinate at output position  
(bh) = y coordinate at output position  
x = the lead address of character string  
y = length of character string

return

(bl), (bh) = next display position is indicated.  
x = address of data shown in last + 1  
y = number of data that was not displayed.  
cy = 0: Displayed all.  
1: Stopped display.

## Setting the type of cursor display (45H)

Set the cursor type.

entry

(cx) = 0000H  
i = 0045H  
a = 0: Cursor is not displayed.  
Bit 7 = 0  
Bit 6 = 0  
Bit 5 = Cursor is displayed.  
Bit 4 = 0  
Bit 3 = Blink  
Bit 2-0 = 

0:	Underline	000
1:	Double underline	001
2:	Full mark	010
3:	Full space	011
4:	Insert mark	100

return

nil

## Symbol display (46H)

Execute ON/OFF of symbol.

entry

(cx) = 0000<sub>H</sub>  
i = 0046<sub>H</sub>

a = symbol pattern (1 = light up)

(bl) = symbol number

- 0: 0, 0, 0, 0, 0, 0, 0, Low battery
- 1: 0, 0, 0, 0, DBL, PRO, RUN, BUSY
- 2: 0, 0, 0, 2ndF, CAPS, hyp, kana, lower case
- 3: 0, 0, 0, DE, G, RAD, E, PRINT

return

nil

## n line scroll-up (47H)

Execute scroll-up. Number of lines can be specified.

entry

(cx) = 0000<sub>H</sub>  
i = 0047<sub>H</sub>

(bx) = 0000<sub>H</sub>

a = number of lines to be scrolled.

return

nil

## n line scroll-down (48H)

Execute scroll-down. Number of lines can be specified.

entry

(cx) = 0000<sub>H</sub>  
i = 0048<sub>H</sub>

(bx) = 0000<sub>H</sub>

a = number of lines to be scrolled.

return

nil

**Clear line (49H)**

Clear the specified line.

entry

(cx) = 0000<sub>H</sub>

i = 0049<sub>H</sub>

(bh) = Y-coordinate of the line to be cleared.

return

nil

**8-dot pattern display (4AH)**

Pattern of one vertical line composed of 8 dots is displayed in down direction from the displayed arbitrary dot.

entry

(cx) = 0000<sub>H</sub>

i = 004A<sub>H</sub>

x = X-coordinate on upper end of the pattern

y = Y-coordinate on upper end of the pattern

a = pattern data (1 = light up, upper end is LSB)

[dotsop] = operation mode when dot is displayed.

*0BFC96H  
(work area)*

0: light up

1: clear

2: reverse

return

nil

**8-dot pattern read (4BH)**

Pattern of one vertical line composed of 8 dots is read in down direction from the displayed arbitrary dot.

entry

(cx) = 0000<sub>H</sub>

i = 004B<sub>H</sub>

x = X-coordinate on upper end of the pattern

y = Y-coordinate on upper end of the pattern

return

a = pattern data (1 = light up, upper end is LSB)

## 1-dot display (4CH)

Displayed arbitrary 1 dot is lit and cleared.

entry

(cx) = 0000H  
i = 004CH  
x = X-coordinate of dot  
y = Y-coordinate of dot  
a = operation  
0: light up  
1: clear  
2: reverse

return

nil

## 1-dot read (4DH)

Displayed arbitrary 1 dot is read.

entry

(cx) = 0000H  
i = 004DH  
x = X-coordinate of dot  
y = Y-coordinate of dot

return

a = data (1 = light up)

## Line display (4EH)

Straight line is drawn between the specified 2 points in accordance with mode.

entry

(cx) = 0000H  
i = 004EH  
x = X-coordinate of start point  
y = Y-coordinate of start point  
(bx) = X-coordinate of end point  
(dx) = Y-coordinate of end point  
[dotsop] = operation mode when dot is displayed.  
0: light up  
1: clear  
2: reverse

0BFC96H  
(work area)

0BFC2AH  
(work area)

return [linptn] = specify the dot pattern in 16-bit data.  
return x = X-coordinate of end point  
y = Y-coordinate of end point  
[linptn] = Pattern next to the last pattern.

### Paint out of box (4FH)

Paint out a rectangle that has diagonal made of the specified 2 points.

entry

(cx) = 0000H  
i = 004FH  
x = X-coordinate of diagonal  
y = Y-coordinate of diagonal  
(bx) = X-coordinate of diagonal  
(dx) = Y-coordinate of diagonal  
[dotstop] = operation mode when dot is displayed.  
0: light up  
1: clear  
2: reverse  
[linptn] = specify the dot pattern in 16-bit data.

return

x = X-coordinate at entry  
y = Y-coordinate at entry  
[linptn] = Pattern next to the last pattern.

### Setting of display state (50H)

ON and OFF of display

entry

(cx) = 0000H  
i = 0050H  
a = display mode  
0: OFF  
1: ON

return

nil

---

### Clearing of display 1 (51H)

Clear the display completely.

entry  
    (cx) = 0000H  
    i = 0051H

return  
    nil

### Clearing of display 2 (52H)

Clear the display from the specified position.

entry  
    (cx) = 0000H  
    i = 0052H  
    a = number of characters to be cleared.  
    (bl) = X-coordinate to start to clear  
    (bh) = Y-coordinate to start to clear

return  
    nil

### n line insertion (54H)

Insert n line.

entry  
    (cx) = 0000H  
    i = 0054H  
    (bh) = Y-coordinate to start insertion  
    a = number of lines you want to insert.

return  
    nil

### Reading of one-line dot pattern (55H)

Expand the dot pattern of one line to an external memory.

entry

(cx) = 0000H

i = 0055H

(bh) = Y-coordinate of line you want to read.

x = data expansion address.

(240 bytes are necessary.)

return

nil

### Writing of one-line dot pattern (56H)

Write (display) the dot pattern of one line on external memory.

entry

(cx) = 0000H

i = 0056H

(bh) = Y-coordinate of line you want to write.

x = data address

return

nil

### Displaying regardless of range (57H)

Display one character regardless of the specified range of display.

entry

(cx) = 0000H

i = 0057H

(bl) = X-coordinate of output position

(bh) = Y-coordinate of output position

a = output data

return

(bl), (bh) = Next display position is indicated.

## Display with guide line (58H)

Character string is displayed with guide line [ ].

entry

(cx) = 0000H  
 i = 0058H  
 (bl) = X-coordinate of output position  
 (bh) = Y-coordinate of output position  
 a = Number of character strings  
 b bit 0 = 0: indicates with [ ]  
 1: indicates with ( )  
 bit 8 = 0: display after clearing the specified line.  
 1: display without clearing the specified line.  
 x = line of character string. One character string ends with FFH.  
 A character string may be composed of any number of characters, but characters over 7 cannot be displayed.  
 When the FEH code is on the lead of one character string, portion succeeding to FEH to the last of that character string is displayed until next FEH appears.  
 Also, when the FEH code is at the location other than the lead of one character string, portion from the lead of character string to the front of it is displayed.

return

nil

## Parameter work

- lcd mode (0BFCA1H, 1 byte)  
 Setting of display mode. Character output succeeding bit 6 is displayed reversely.
- lcd crsr x (0BFC9BH, 1 byte)  
 X-coordinate of cursor
- lcd crsr y (0BFC9CH, 1 byte)  
 Y-coordinate of cursor
- lcd width (0BFC9DH, 1 byte)  
 Number of digits possible to display is shown.
- lcd height (0BFC9EH, 1 byte)  
 Number of lines possible to display is shown.
- scrn crsr x (0BFC27H, 1 byte)  
 X-coordinate displayed next to stdo :scrn :device is shown.

- scrn srst y (0BFC28<sub>H</sub>, 1 byte)  
Y-coordinate displayed next to stdo: /scrn: device is shown.
- dotsop (0BFC96<sub>H</sub>, 1 byte)  
Specify operational method of the dot that has been already displayed by the line etc. combined with the dot that is about to be displayed.
  - 0: light up
  - 1: clear
  - 2: reverse
- linprn (0BFC2A<sub>H</sub>, 2 bytes)  
Specify the dot pattern that is displayed by the line or boxfull with 16 dots.
- (0BFC29<sub>H</sub>, 1 byte)  
Control code succeeding bit 6 is displayed.  
(No control operation is executed.)
- Storing address of character font
  - (BFC87<sub>H</sub>, 3 bytes) Storing address for character font 00<sub>H</sub>-1F<sub>H</sub>
  - (BFC90<sub>H</sub>, 3 bytes) Storing address for character font 20<sub>H</sub>-7F<sub>H</sub>
  - (BFC8A<sub>H</sub>, 3 bytes) Storing address for character font 80<sub>H</sub>-9F<sub>H</sub>
  - (BFC93<sub>H</sub>, 3 bytes) Storing address for character font A0<sub>H</sub>-CF<sub>H</sub>
  - (BFC8D<sub>H</sub>, 3 bytes) Storing address for character font E0<sub>H</sub>-F0<sub>H</sub>One character is composed of 6 bytes.

## No. 01 Key Driver (STDI: KYBD:)

The key driver is a device of the following type.

- Device possible to use as a file.
- Standard character device.
- Only read is possible.
- The key driver is opened as the standard input of STDI:

### Command list

- Command for standard character device
 

08 <sub>H</sub>	<del>No processing</del>	Non-operation
09 <sub>H</sub>	<del>No processing</del>	Non-operation
0A <sub>H</sub>	Key input	
0B <sub>H</sub>	Error	
0C <sub>H</sub>	Key input	
0D <sub>H</sub>	Error	
0E <sub>H</sub>	Key input	
0F <sub>H</sub>	Error	
- Command for standard block device
 

10 <sub>H</sub> -1F <sub>H</sub>	Error
----------------------------------	-------
- Command for special device
 

20H-2FH	Error
---------	-------
- Command peculiar to each device
 

3F <sub>H</sub>	<i>Formatting</i>
40 <sub>H</sub>	Initialization of each parameter
41 <sub>H</sub>	Read out matrix code
42 <sub>H</sub>	Non-destructive read out of matrix code
43 <sub>H</sub>	Key read out
44 <sub>H</sub>	Set data to key buffer
45 <sub>H</sub>	Buffer clear
46 <sub>H</sub>	Display of function key

### Format (3F<sub>H</sub>)

Return the pointer of the table that converts matrix code to ASCII code. (The pointer is not referred inside the device driver.)

```
entry
  (cx) = 0001H
  i = 3FH
```

```
return
  nil
```

Pointer of conversion table (each 3 bytes)

- [0BFC2DH]: for 1-byte code
- [0BFC30H]: for 2-byte code
- [0BFC33H]: for 1-byte code of SHIFT
- [0BFC36H]: for 2-byte code of SHIFT
- [0BFC39H]: for 1-byte code of CTRL
- [0BFC3CH]: for 2-byte code of CTRL

62 bytes per one table

### Initialization of key device driver (40H)

entry

(cx) = 0001H  
 i = 0040H  
 a = level of initialization  
 0: all reset  
 1: reset  
 2: off  
 3: on

return

nil

### Read out of matrix code (41H)

non-“BUSY”  
 (cpu is stopped)

Next data is taken out of the buffer of the key scan.

If the key is not input, next key input is waited in the state of low power. In case key scan is impossible by some reason, such as low battery etc., return as cy = 1.

When this routine is called up, counter for auto power off is reset and activates decrement. If the key is not input even if the counter is showing 0, code 0FH is brought back.

entry

(cx) = 0001H  
 i = 0041H

return

cy = 0

a: bit 7 = 1: when key is released.

0: when key is pressed.

bits 0-6 = Matrix code

b = 0: no key input

1: key input

cy = 1 error (cancel.)

a = error code

### Non-destructive reading of matrix code (42H)

The next data is read from the buffer of key scan. However, the data is not removed from the buffer. In case key is not input, return immediately.

entry

(cx) = 0001H  
i = 0042H

return

cy = 0

a: bit 7 = 1: when key is released.

0: when key is pressed.

bits 0-6 = Matrix code

b = 0 : no key input

1 : key input

cy = 1 : error

a = error code

### Matrix code table

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		0	SPACE	P	hyp	RCL									
1	shift	1	A	Q	sin	STO									
2	ctrl	INS	2	B	R	cos									
3	ON/BRK	F1	3	C	S	tan									
4	OFF	F2	4	D	T	FSE									
5	BASIC	F3	5	E	U	HEX									
6	MENU	F4	6	F	V	DEG									
7	カナ	F5	7	G	W	In									
8	BS		8	H	X	Int									
9	CAPS		9	I	Y	1/x									
A	2nd F		*	J	Z	↓									
B	DEL		+	K	=	EXP									
C	C-CE	▶	.	L	:	y <sup>x</sup>									
D	◀	◆	-	M	(	√									
E	↑	↓	/	O	)	x <sup>2</sup>									
F	↑	↓	/	O		+/-									
	Auto power OFF														

## Key reading (43H)

Key input routine for the Roman character conversion routine. Execute processing of shift key and ctrl key and then return in 2 bytes of shift JIS + expansion code. In case of no key at call time, it is possible to specify either of waiting for key input or executing immediate return.

Japanese language

entry

(cx) = 0001H

i = 0043H

a:bit 7

- 0: return immediately when key does not exist.
- 1: wait for next key in low power mode when key does not exist.

return

i = 0: No key input

1: key input

a = key data of the first byte

b = key data of the second byte

cy = 1: error

a = error code

example)

	Input	①	②
Keyed-in data	[ A ]	[CTRL] + [off]	
1st byte	41H	00H	
2nd byte	00H	0FH	

NEXT PAGE!

Value returned by key reading ① The first byte a = ?? (value in the table)  
b = 00H

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
0	*	CTRL P	SPACE	0	@	P	'	p			-	タ	ミ		
1	CTRL A	CTRL Q	!	1	A	Q	a	q			.	ア	チ	ム	
2	CTRL B	CTRL R	#	2	B	R	b	r			フ	イ	ツ	メ	
3	CTRL C	CTRL S	^	3	C	S	c	s			」	ウ	テ	モ	
4	ON BRK										,	エ	ト	ヤ	
5	CTRL D	CTRL T	\$	4	D	T	d	t			、	オ	ナ	ユ	
6	CTRL E	CTRL U	%	5	E	U	e	u			・	カ	ニ	ヨ	
7	CTRL F	CTRL V	&	6	F	V	f	v			ヲ	キ	ヌ	ラ	
8	CTRL G	CTRL W	.	7	^G	W	g	w			ア	キ	ヌ	ラ	
9	CTRL H	CTRL X	(	8	H	X	h	x			イ	ク	ネ	リ	
A	CTRL I	CTRL Y	)	9	I	Y	i	y			ウ	ケ	ノ	ル	
B	CTRL J	CTRL Z	*	:	J	Z	j	z			エ	コ	ハ	レ	
C	CTRL K	CTRL =	+	:	K	[	k	l			オ	サ	ヒ	ロ	
D	CTRL L	C CE	▶	.	<	L	¥	i			ヤ	シ	フ	ワ	
E	CTRL M	◀	-	=	M	]	m	l			ユ	ス	ヘ	ン	
F	CTRL N	↑	.	>	N	^	n	-			■	セ	ホ	・	
G	CTRL O	↓	/	?	O	-	o	DEL			フ	ソ	マ	・	

## Data set to keyboard buffer (44H)

Set the data to keyboard buffer.

Set the keyed-in data displayed with data type of shift JIS + expansion code in the buffer in which result of Roman character conversion or contents of key function is. Current data in the buffer is lost.

entry

(cx) = 0001H  
i = 0044H  
x = lead address of set data  
a = number of set data bytes

return

nil

## Key clear (45H)

Clear the key scan buffer and keyboard buffer.

entry

(cx) = 0001H  
i = 0045H

return

nil

Value returned by key reading ② the second byte a = 00, b = ?? (value in the table)

Display the following key in 2 bytes by the code came immediately after 00H code.

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
CTRL hyp					CTRL 0	CTRL SPACE		RCL		SHIFT RCL		CTRL RCL		
					CTRL 1	CTRL :	sin	STO	sin <sup>-1</sup>	SHIFT STO	CTRL sin	CTRL STO		
2:			SHIFT INS	CTRL INS	CTRL 2	CTRL (	cos		cos <sup>-1</sup>	→xy	CTRL cos			
			SHIFT ON BRK	CTRL ON/BRK	CTRL PF1	CTRL 3	CTRL )	tan		tan <sup>-1</sup>	nl	CTRL tan		
OFF		SHIFT OFF			CTRL PF2	CTRL 4		FSE	sin h	TAB	sin h <sup>-1</sup>	CTRL FSE		
5:	BASIC	AER	CTRL BASIC	CTRL PF3	CTRL 5			→HEX	cos h	→DEC	cos h <sup>-1</sup>	CTRL HEX		
6:	menu	CAL	CTRL menu	CTRL PF4	CTRL 6			→DEG	tan h	→D.MS	tan h <sup>-1</sup>	CTRL DEG		
7:	カナ	SHIFT カナ	CTRL カナ	CTRL PF5	CTRL 7			In		e <sup>+</sup>		CTRL In		
8:		SHIFT BS	CTRL BS		CTRL 8			log		10 <sup>+</sup>		CTRL log		
9:	CAPS	SHIFT CAPS	CTRL CAPS		CTRL 9			1/x		→πθ		CTRL 1/x		
A:					CTRL *		↓			SHIFT I		CTRL I		
B:		SHIFT DEL	CTRL DEL		CTRL +			EXP		π		CTRL EXP		
C:		CA	SHIFT CA	CTRL C.CE	CTRL ▶	CTRL C.CE	◀	y <sup>n</sup>		<sup>n</sup> √y		CTRL y <sup>n</sup>		
D:		SFIFT	SHIFT SFIFT	CTRL SFIFT	CTRL ▲	CTRL ▼		√		<sup>3</sup> √		CTRL ∛		
E:	◆	DEG	SHIFT ◆	SHIFT DEG	CTRL ◆	CTRL DEG		x <sup>2</sup>		%		CTRL %		
F:	CTRL OFF		SHIFT ↓		CTRL ↓	CTRL /		+/-		SHIFT +/-		CTRL +/-		

= Auto power OFF

## Display of function key (46H)

Display or clear the function key. Display status must be below 3 lines. And, if bit 2 of [BFC3FH] is set at "1", display is prohibited.

entry

(cx) = 0001H  
i = 0046H

return

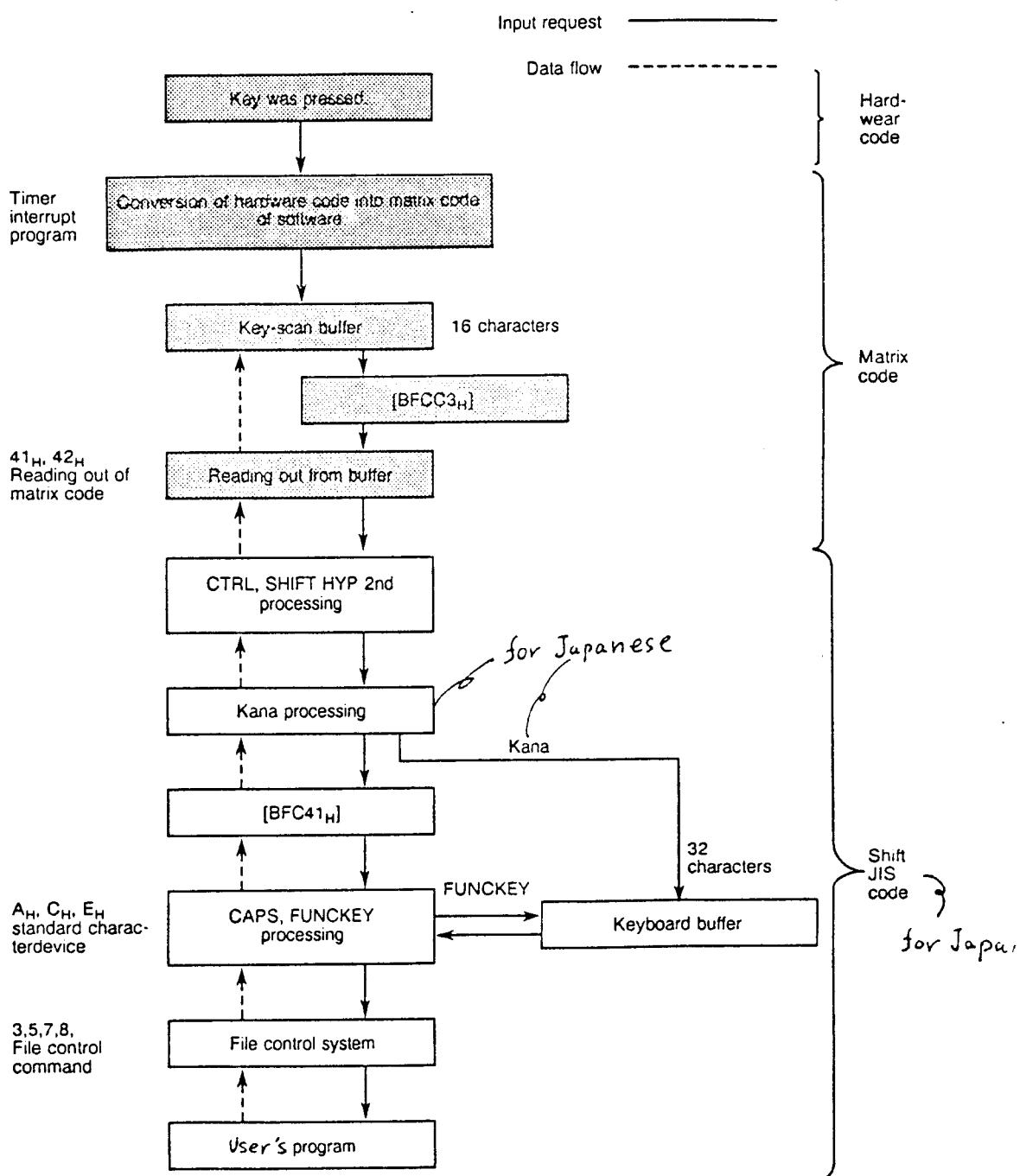
cy = 1 It was in 4-line display state.

## Parameter for key input device driver

- repeat wait (0BFCBA<sub>H</sub>, 1 byte)  
Specify the time from pressing of key until start of repeat. Unit: 16 msec.
- repeat pitch (0BFCBB<sub>H</sub>, 1 byte)  
Interval between two keys in repeat state. Unit: 16 msec
- auto power off time (0BFCBC<sub>H</sub>, 2 bytes, Low, hi)  
Time for auto power off. Unit: approx. 0.5 sec
- 0BFCBE<sub>H</sub> [SOFTINT]  
bit 7: break  
bit 6: low battery
- 0BFCBF<sub>H</sub>  
bit 7: repeat on  
bit 4: click on
- (BFCC0<sub>H</sub>, 3 bytes)  
Lead address of the table to convert the code in hardware into the matrix code in software.  
This table corresponds to key matrix of hardware. Key can be rearranged.
- (BFCC3<sub>H</sub>, 3 bytes) Both are hook for key processing routine.  
(BFC41<sub>H</sub>, 3 bytes)
- (BFC45<sub>H</sub>, 1 byte)  
Slot number having function key data (FUNCKEY.).  
Ex: If it is 0, item in "S1:" is referred.
- (BFC46<sub>H</sub>-BFC51<sub>H</sub>, 9 bytes)  
File name having function key data.

- Internal RAM FF<sub>H</sub>  
bit 3: break key

### From key pressing until reading out with file control system



---

## No. 02 SIO (RS232C) Driver (COM:)

SIO driver is the device of following type.

- Device usable as a file.
- Special device
- Read/write possible.

### Command List

- Command for standard character device  
 $08_{H}$ - $0F_{H}$  error
- Command for standard block device  
 $10_{H}$ - $1F_{H}$  error
- Command for special device  
 $29_{H}$ ,  $2B_{H}$ ,  $2D_{H}$ ,  $2E_{H}$ ,  $2F_{H}$  error  
 $20_{H}$ - $2F_{H}$  excluding the above supported  
However, setting of a register is ignored at reading block ( $25_{H}$ ). This is the same for verify ( $27_{H}$ ). Use byte reading.
- Command peculiar to each device
  - $3F_{H}$  Format
  - $40_{H}$  Initialization of each parameter
  - $41_{H}$  Direct output of 1 byte
  - $42_{H}$  Direct input of 1 byte
  - $43_{H}$  Setting the hardware
  - $44_{H}$ - $49_{H}$  Setting of RSL, RR, and ER ports
  - $4A_{H}$ - $4B_{H}$  Reading of CS and CD ports

### Format ( $3F_{H}$ )

Clear the parameter and buffer to return to initial state of entire no input.

entry  
(cx) =  $0002_{H}$   
i =  $3F_{H}$

return  
nil

---

### Initialization of each parameter (40H)

```
entry
(cx) = 0002H
i = 40H
a = level of initialization
  0: all reset
  1: reset
  2: off
  3: on
```

### 1-byte direct output (41H)

Output 1-byte data without conversion. However, X-control is possible according to the setting condition.

```
entry
(cx) = 0002H
i = 41H
a = output data

return
cy = 0: no error
cy = 1: error occurred
a = 00H: time out
      FEH: low battery
      FFH: break
```

### 1-byte direct input (42H)

Input 1-byte data directly. X-on (11H) and X-off (13H) controls are ignored. X-control is possible according to the setting condition.

```
entry
(cx) = 0002H
i = 42H

return
cy = 0: no error
a = input data
cy = 1: error
a = 00H: time out
      FEH: low battery
      FFH: break
```

### Setting of hardware (43H)

Set the condition of IOCS system work in hardware and format.

entry  
(cx) = 0002H  
i = 43H

return  
nil

### Setting of RS, RR, and ER ports (44H-49H)

entry  
(cx) = 0002H  
i = 44H: Set RS port at high level.  
45H: Set RS port at low level.  
46H: Set RR port at high level.  
47H: Set RR port at low level.  
48H: Set ER port at high level.  
49H: Set ER port at low level.

return  
nil

### Reading of CS and CD ports (4AH-4BH)

entry  
(cx) = 0002H  
i = 4AH: Specify CS port.  
4BH: Specify CD port.

return  
cy = 0: Port is specified at low level.  
1: Port is specified at high level.

## Parameter Work

- SIO timer master 0BFD31<sub>H</sub>-0BFD32<sub>H</sub>  
Time n on error timer × 0.5 (sec) However, 0FFFF<sub>H</sub> is unlimited.  
default value = 0FFFF<sub>H</sub> (unlimited)
- SIO baud rate 0BFD33<sub>H</sub>  
Specify baud rate, length, parity  
default value = 3C<sub>H</sub> (00111100<sub>B</sub>)

	Bit 6, 5, 4 Baud rate		Bit 3, 2 Parity		Bit 1 Character length (Pdata)		Bit 0 Stop bit	
000	None	00	Even-parity		0: 8 bits		0:	1 bit
001	300 baud	01	Odd-parity		1: 7 bits		1:	2 bits
010	600 baud	10	Non-parity					
011	1200 baud	11	Non-parity					
100	2400 baud							
101	4800 baud							
110	9600 baud							

### Example

3C<sub>H</sub>: 1200 baud, Non-parity, 8-bit length, stop 1bit

### Note

If the specified condition has been changed, '43<sub>H</sub>' hardware set IOCS must be called.

Bit 7: 0

- SIO Setup 0BFD34<sub>H</sub>  
Specify shift in/out, X on/off. Specify transfer of transmission code at open/close.  
Default value = 21<sub>H</sub>
  - Bit 6 = 0: 1-byte data stored in 'SIO open send data' is not transmitted at open state.  
1: Transferred 'SIO open send data' = 0BFD61<sub>H</sub>
  - Bit 4 = 0: 1-byte data stored in 'SIO close send data' is not transmitted at close state.  
1: Transferred 'SIO close send data' = 0BFD62<sub>H</sub>
  - Bit 2 = 0: Without X-on/off designation at receiving  
1: With designation
  - Bit 1 = 0: Without X-on/off designation at sending  
1: With designation

Bit 0 = 0: Without shift in/out designation  
1: With designation

- SIO receive port condition 0BF0D35<sub>H</sub>  
Control of receive port  
Default value = 02<sub>H</sub>  
Bit 2 CS = 0: don't care  
1: take in as receiving data when the CS signal is high and ignore at low.  
Bit 1 CD = 0: don't care  
1: take in as receiving data when the CD signal is high and ignore at low.
  
- SIO receive port control 0BF0D36<sub>H</sub>  
Control of receive port  
Default value = 0DF<sub>H</sub>  
Bit 6 ER = 0: When receiving buffer becomes full, ER signal becomes low.  
1: don't care  
Bit 5 RR = 0: When receiving buffer becomes full, RR signal becomes low.  
1: don't care  
Bit 4 RS = 0: When receiving buffer becomes full, RS signal becomes low.  
1: don't care

*Example*

0DF<sub>H</sub>

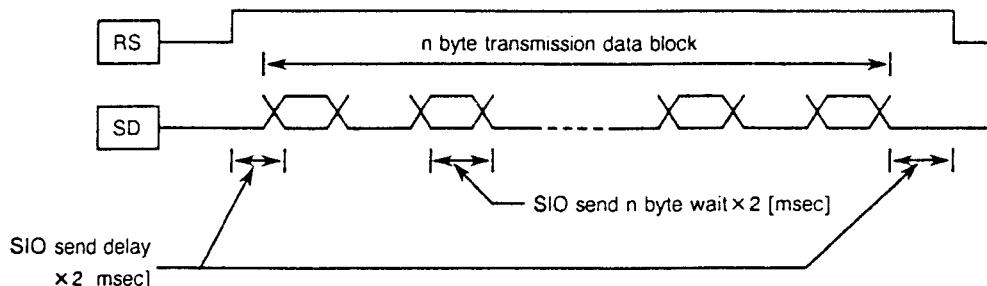
If RR-receiving buffer becomes full, the transmission side is stopped in RR Low.

- SIO send port condition 0BF0D37<sub>H</sub>  
Control of SEND port  
Default value = 04<sub>H</sub>  
Bit 2 CS = 0: don't care  
1: Transmit when the CS signal becomes high.  
When the CS signal is low, wait until it becomes high.  
(within range of error timer)  
Bit 1 CD = 0: don't care  
1: Transmit when the CD signal becomes high.  
When the CD signal is low, wait until it becomes high.  
(within range of error timer)
- ↳ BF0D31<sub>H</sub> ~ 32<sub>H</sub>
- SIO send port control 0BF0D38<sub>H</sub>  
Control of send port  
Default value = 50<sub>H</sub>  
Bit 6 ER = 0: don't care.  
1: ER signal becomes high before transfer of transmission data block and becomes low after transfer.  
Bit 5 RR = 0: don't care.  
1: RR signal becomes high before transfer of transmission data block and becomes low after transfer.

Bit 4 RS = 0: don't care.

- 1: RS signal becomes high before transfer of transmission data block and becomes low after transfer.

- SIO send delay 0BF3D9H  
[00-0FFH] × 2 [msec] wait time is specified before or after transmission data block at transmission.  
Default value = 01H [ 2 msec ]



- SIO crlf 0BF3D9H  
Specify the delimiter. External code is converted into internal delimiter (0DH + 0AH)  
Default value = 01H      cr lf
- Bit 1, 0: 00 = Not use  
01 = 0DH  
10 = 0AH  
11 = 0DH + 0AH

#### Example

when specified the delimiter at 0DH with 01, code is converted as follows.

External	Internal E500
at sending    0DH + 0AH → 0DH	0DH      ← 0DH + 0AH
at receiving    0DH → 0DH + 0AH	0DH      → 0DH + 0AH

- SIO eof code 0BF3DCH  
To specify the end code (external code is converted in internal end code (1AH).)  
Default value = 1AH

#### Example

When specified in end code 09H, code is converted as follows.

External	Internal E500
at sending    1AH → 09H	09H      ← 1AH
at receiving    09H → 1AH	09H      → 1AH

- SIO open close wait 0BFD40H  
Wait  $n \times 0.5$  (msec) immediately after opening or immediately before closing.  
(for level converter CE-130T standby)  
Default value = 04H (20 msec)

*Note*      *maximum*  
Input of 00H specifies  $256 \times 5$  (msec) .... minimum 01H

- SIO open port control 0BFD41H  
Open of SIO port.  
Default value = 41H  
Bit 6 ER = 0: don't care  
    1: ER signal becomes high at open and low at close.  
Bit 5 RR = 0: don't care  
    1: RR signal becomes high at open and low at close.  
Bit 4 RS = 0: don't care  
    1: RS signal becomes high at open and low at close.
- SIO send n byte wait 0BFD60H  
Specify insertion time of  $[00 \times OFF_H] \times 2$  (msec) wait between send data 1 byte at sending.  
Default value = 00H (non-wait)
- SIO open send data 0BFD61H  
Default value = 11H  
When 'SIO setup' bit-6 is 1, SIO open send data is transferred by 1 byte at open.
- SIO close send data 0BFD62H  
Default value = 13H  
When 'SIO setup' bit-4 is 1, SIO close send data is transferred by 1 byte at close.

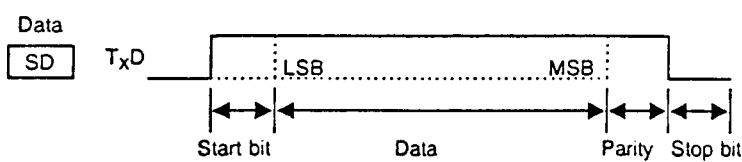
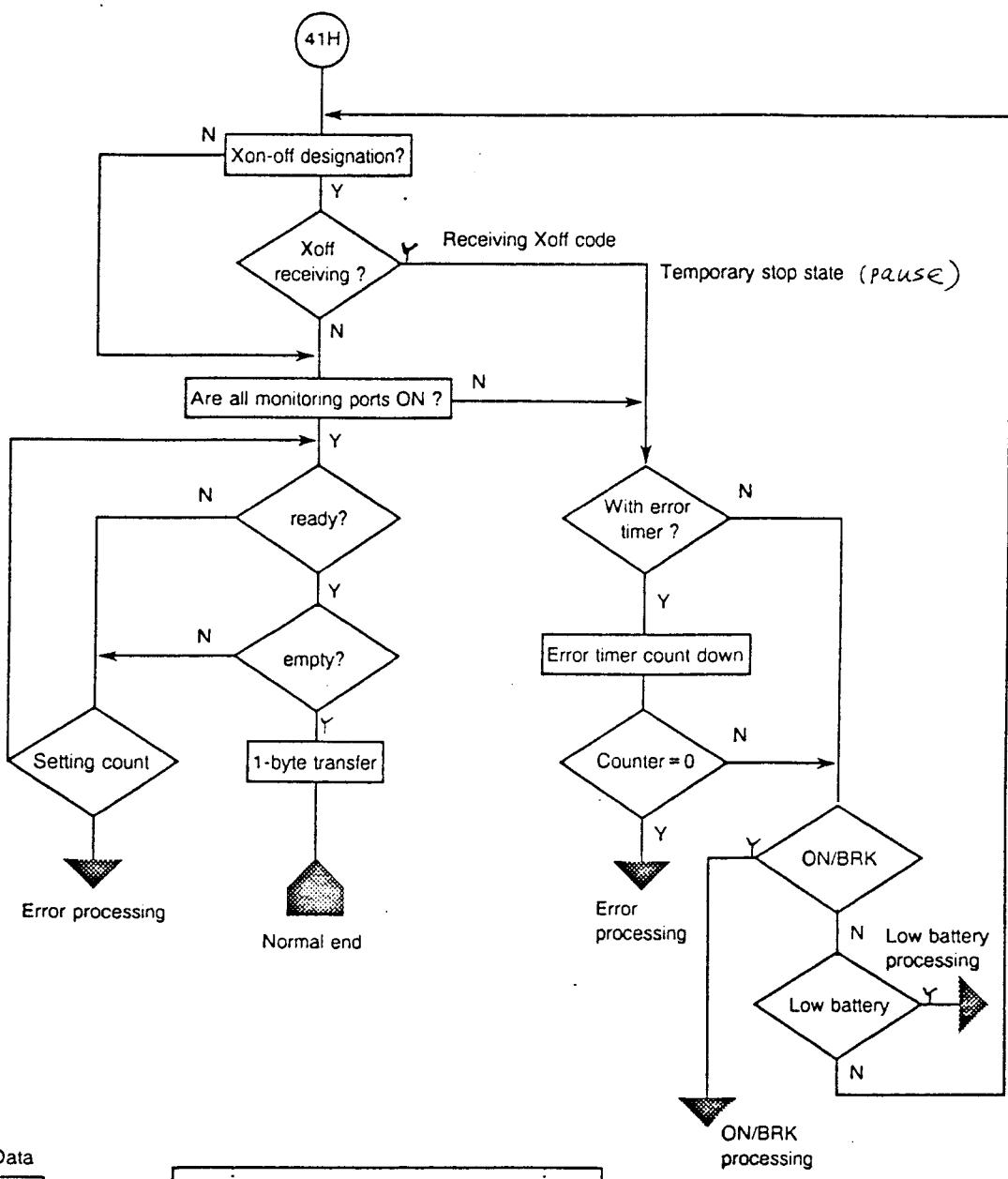
#### SIO hardware specifications

Baud rate: 300, 600, 1200, 2400, 4800, 9600 (bps)  
Data length: 7 or 8 bits  
Stop bit : 1 or 2 bits  
Parity : even number, odd number, none  
Duplex  
Xon-off : possible to specify  
Shift in/out: possible to specify

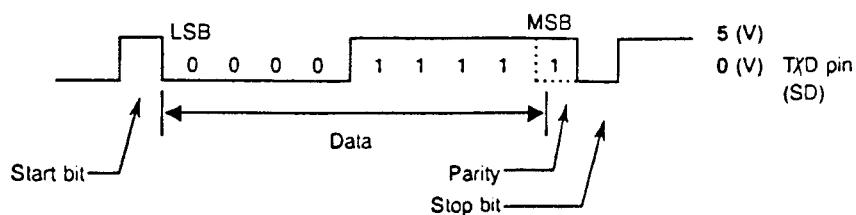
#### SIO software specifications

##### Transmission (1 byte)

In case of Xon-off is specified, if Xoff code is being received, transmission side keeps waiting until X-code is received and released.  
And, if the signal (port specified with 'SIO send port condition') to be monitored is not set ON (high level), it keeps waiting.  
When above conditions are satisfied and in CPU ready, empty state, 1-byte data is output.

**Example**

data=F0H , parity odd number, stop bit 1



X-off code	13H	at sending:	When Xoff code is received, sending stops temporarily.
		at receiving:	If the buffer is almost full, X-off code is sent and transmission is stopped temporarily.
X-on code	11H	at sending:	When transmission is stopped temporarily, if Xon code is received, transmission is resumed.
		at receiving:	If the buffer is empty and the opponent side is temporarily stopped, Xon code is sent and then receiving is resumed.
※ shift-in code : 0FH		shift-out code : 0EH	

#### Reception (1 byte)

PC-E500 has the receiving buffer for reception of data from SIO. When the data is being received, SIO reception is interrupted and interrupt routine writes the data. Xon and Xoff control codes do not write in the receiving buffer.

SIO buffer is the ring buffer controlled by write-pointer and read-pointer. When the received data are accumulated and the receiving buffer come short, Xoff code is transferred if specified by Xon-off. Also, regardless whether specified or not, the port that is indicated by SIO receive port control is made off (low) status and the unit of other party is stopped.

When buffer becomes empty, Xon code is transferred. And regardless whether specified or not by Xon-off, the port that is indicated by SIO receive port control is made on (High) status and permission of transfer is given to the unit of other party.

When the data is read from the receiving buffer, it is checked whether the data is in the receiving buffer. If there is the data, 2 bytes is read.

Pin number	Signal name	Symbol	Signal direction	Functions
2	Send Data	SD (Tx <sub>D</sub> )	Output	Data signal to be sent
3	Receive Data	RD (Rx <sub>D</sub> )	Input	Receiving data signal
4	Request to Send	RS (RTS)	Output	This signal become high level by data transmission and low level by transmission end.
5	Clear to Send	CS (CTS)	Input	When data is sent, transmission is executed if this signal is in high level, and is stopped if the signal is in low level.
7	Signal Ground	SG		Adjust the reference electric potential between input/output devices.
8	Carrier Detect	CD	Input	Transmission is executed when this signal is in high level and is stopped when the signal is in low level.
11	Receive Ready	RR	Output	High level when reception is possible and low level when reception is impossible.
14	Equipment Ready	ER (DTR)	Output	When serial input/output device circuit is open (if executed OPEN command), signal becomes high level.
1	Frame ground	FG		Grounding for maintenance
10 13		VC		Supply voltage

Notes 1) High level means voltage level of VC. Low level means voltage level of SG.  
 2) As the inside is composed of C-MOS parts, if voltage exceeding permissible range (voltage level between SG-VC) is given to input/output pin, the inside may be broken.

## No. 03 Printer Driver (STDL:PRN:)

This device is explained as follows.

- Available as a file.
- Standard character device
- Only write is possible.
- STDL : Printer driver is opened as the standard listing output.

### Command list

- Standard character device command
  - $08_H$  open
  - $09_H$  close
  - $0A_H$  error
  - $0B_H$  output to printer
  - $0C_H$  error
  - $0D_H$  output to printer
  - $0E_H$  error
- Standard block device command
  - $10_H$ - $1F_H$  error
- Command for special block device
  - $20_H$ - $2F_H$  error
- Command peculiar to each device
  - $3F_H$  Format
  - $40_H$  Initialization of each parameter
  - $41_H$  Printing data output
  - $42_H$  Read of printing position
  - $43_H$  Printer check

### Format ( $3F_H$ )

entry  
(cx) =  $0003_H$   
i =  $3F_H$

return  
nil

**Initialization of each parameter (40H)**

entry  
(cx) = 0003H  
i = 40H  
a = level of initialization  
0: all reset  
1: reset  
2: off  
3: on

**Printing data output (41H)**

entry  
(cx) = 0003H  
i = 41H  
a = output data

return  
cy = 0: no error  
= 1: error

**Read of printing position (42H)**

entry  
(cx) = 0003H  
i = 42H

return  
a = position of present printing head

**Printer check (43H)**

entry  
(cx) = 0003H  
i = 43H

return  
cy = 0: a = type of printer  
i = number of max. printing digit  
cy = 1: not connected  
a = 54H (Printer error)

---

## No. 04 Tape Driver (CAS:)

Tape driver is the device driver shown as follows.

- Device can be used as a file.
- Special device
- Both of read and write possible. Execute read only or write only one time.

### Command List

- Standard character device command  
 $08_{H}$ - $0F_{H}$  error
- Standard block device command  
 $10_{H}$ - $1F_{H}$  error
- Special device command  
 $29_{H}$ ,  $2B_{H}$ ,  $2D_{H}$ ,  $2E_{H}$ ,  $2F_{H}$  error  
 excluding the above command  $20_{H}$  -  $2FH$  supported
- Command peculiar to each device  
 $3F_{H}$  format  
 $40_{H}$  initialization of each parameter  
 $41_{H}$ - $43_{H}$  write, read, verify of data block  
 $44_{H}$  -  $45_{H}$  write, read of the header block

### Write, read, and verify of the header block ( $44_{H}$ - $45_{H}$ )

#### entry

(cx) =  $0004_{H}$   
 i =  $44_{H}$ : Output to tape  
 =  $45_{H}$ : Input from tape  
 x = address of header block

#### return

cy = 0: no error  
 x = next address of the data that could be transferred  
 (compared).  
 y = number of bytes of the data that could be transferred  
 (compared).  
 cy = 1: error  
 x = next address of the data that was correctly transferred  
 (compared).  
 y = number of bytes of the data that was correctly  
 transferred  
 a = error code

**Initialization of each parameter (40H)**

```

entry
(cx) = 0004H
i = 40H
a = level of initialization
  0: all reset
  1: reset
  2: off
  3: on

return
= nil

```

**Write, read, and verify of the data block (41H-43H)**

```

entry
(cx) = 0004H
i = 41H: Output to tape
        42H: Input from tape
        43H: Input from tape and comparison (verify)
x = data address
y = data size

return
cy = 0: no error
      x = next address of the data that could be transferred
          (compared).
      y = number of bytes of the data that could be transferred
          (compared).
cy = 1: error
      x = next address of the data that was correctly transferred
          (compared).
      y = number of bytes of the data that was correctly
          transferred
      a = error code

```

**Format (3FH)**

All parameter are initialized.

```

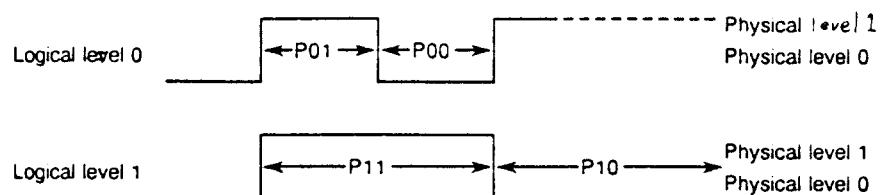
entry
(cx) = 0004H
i = 3FH

return
nil

```

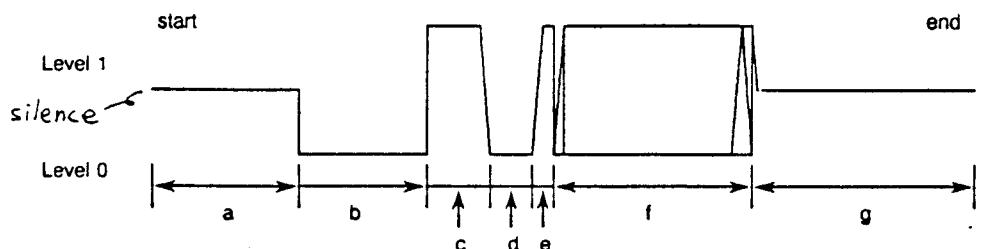
## Parameter work

- Pulse length (baud rate can be changed.)
  - (BFD42H, 1 byte) length of logical level 0 and physical level 1 (P01)
  - (BFD43H, 1 byte) length of logical level 0 and physical level 0 (P00)
  - (BFD44H, 1 byte) length of logical level 1 and physical level 1 (P11)
  - (BFD45H, 1 byte) length of logical level 1 and physical level 0 (P10)
  - (BFD46H, 1 byte) threshold of logical level 0 and physical level 1



Following 0 and 1 should be logical level.

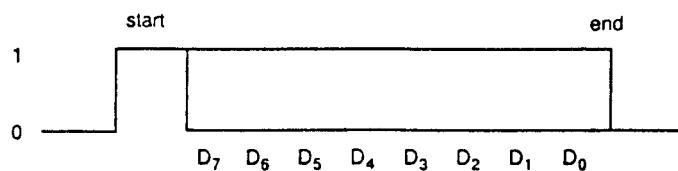
- Header block and data block



	a	b	c	d	e	f	g
Header block at write	[BFD48H]	[BFD4A <sub>H</sub> 2 byte]	[BFD4C <sub>H</sub> ]	[BFD4D <sub>H</sub> ]	1	30 byte + $\alpha$	[BFD49H]
Header block at read	more than '0'	more than [BFD53H, 2 byte]	[BFD4C <sub>H</sub> ]	[BFD4D <sub>H</sub> ]	1	30 byte + $\alpha$	more than '0'
Data block at write	0	[BFD4A <sub>H</sub> 2 byte]	[BFD50H]	[BFD51H]	1	0 or more arbitrary byte + $\alpha$	[BFD52H]
Data block at read	more than '0'	more than [BFD53H, 2 byte]	[BFD50H]	[BFD51H]	1	0 or more arbitrary byte + $\alpha$	more than '0'

$\alpha$  : Check sum (1 byte)

- Structure of 1 byte



### Structure of file

One file consists of one header block and some data blocks.  
Data of header block is always  $30H$  bytes with following composition.

	+ 0	+ 2	+ 4	+ 6	+ 8	+ A	+ C	+ E
+ 00	04	File name			20 20 20 20 20	20	Expansion	
+ 10	→ 0D	00 00 00 00 00 00	04	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
+ 20	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

The most top is the discriminator.  $04H$  indicates that this is the file via 'file control system'(ex. SAVE, OPEN commands, etc.). Files created by csave, csavem do not take this form.

Basically, length of the data block is arbitrary. One data block created by open or save is 256 bytes. In this case, file ends with  $1AH$ .

- Structure of CSAVE header block

	+ 0	+ 2	+ 4	+ 6	+ 8	+ A	+ C	+ E
+ 00	02	File name			20 20 20 20 20	20 20 20 20 20	20 20 20 20 20	
+ 10	20	0D	DI	Dh	00 00 00 00 00 01	00 00 00 00 00 00	De	00 00
+ 20	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

(DI, Dh, De) = Data size

- Structure of CSAVE header block

	+0	+2	+4	+6	+8	+A	+C	+E
+00	01	File name						20 20 20 20 00 00
+10	00	0D	DI	Dh	SI	Sh	EI	Eh 00 00 00 00 00 De Se Ee
+20	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00 00

(DI, Dh, De) = data size  
(SI, Sh, Se) = data start address  
(EI..Eh, Ee) = entry address

## No. 05 RAM Disk Driver (E: F: G:)

This driver is the device of following type.

- Device that can be used as a file.
- Standard block device
- Read/write possible

### Command List

- Standard character device command  
 $08_{H}$ - $0F_{H}$  Error
- Standard block device command  
 $10_{H}$ - $17_{H}$  Supported
- Special device command  
 $20_{H}$ - $2F_{H}$  Error
- Command peculiar to each device  
 $3F_{H}$  Format  
 $40_{H}$  Initialization of each parameter

### Format ( $3F_{H}$ )

(f/e)

Secure, change and release the RAM disk area. The RAM disk area is secured as memory block "RAMFILE." as E on S1, F on S2 and G on S3.

```

entry
    (cl) =  $05_{H}$ 
    (ch) = drive number (0/1/2)
        i =  $3F_{H}$ 
        x = lead address of set information character

return
    cy = 0: no error
        x = last of the set information + 1
    cy = 1: error
        a = same error number as BASIC

```

Initialization of each parameter (40H)

entry

(cx) = 0005H  
i = 40H  
a = level of initialization  
0 = all reset  
1 = reset  
2 = off  
3 = on

return

nil

---

## No. 06 Memory Block Driver (S1: S2: S3)

This driver is the device of following type.

- Device that can be used as a file.
- Special device
- Read/write possible

### Command list

- Standard character device command  
 $08H-0FH$  Error
- Standard block device command  
 $10H-1FH$  Error
- Special device command  
 $20H-2FH$  Supported
- Command peculiar to each device
 

$3FH$	Format
$40H$	Initialization of each parameter
$41H$	Search for physical address
$42H$	Change of block size
$43H$	Transfer of block
$44H$	Rename of block
$45H$	Creation of memory block
$46H$	Deletion of memory block
$47H$	Condense
$48H$	Create memory block on the block top. (not supported)

### Initialization of memory driver ( $40H$ )

```

entry
  (cx) = 0006H
  il = 40H
  a = level of initialization
    0: all reset
    1: reset
    2: off
    3: on

```

```

return
  nil

```

### Format (3FH)

Connect and disconnect RAM in the pocket computer with RAM in the RAM card.

entry  
(cx) = 0006H  
i = 3FH  
x = lead address of the set information character string  
[x] = "P": connection. Combined one becomes S1:  
= "S": disconnection. Divided into "S" S1: and S2:  
  
return:  
x = next address  
cy = 0: no error  
1: error  
a = 0AH: syntax error  
3CH: memory error

### Search for physical address (41H)

Search for the lead address of the block from the memory block name.

entry  
(cl) = 06H  
(ch) = slot number (0/1/2)  
i = 41H  
x = address of block name character string  
  
return  
cy = 0: no error  
x = last of block name + 1  
y = lead address of block  
cy = 1: error  
x = (no change)  
a = 01H: no specified slot.  
04H: no block is found.

### Condense (47H)

Fill the space in each memory block.

entry  
(cl) = 06H  
(ch) = slot number  
i = 47H  
  
return  
nil

**Change of block size (42H)**

```

entry
  (cl) = 06H
  (ch) = slot number
  i = 42H
  a = free area pointer number (0/1)
  x = address of block name character string
  y = request size

return
  cy = 0: no error
  x = last of block name + 1
  cy = 1: error
    x: (no change)
    a = 00H: slot (card) is protected.
            01H: no specified slot.
            04H: block is not found.
            0CH: insufficient memory
            y = size possible to change
            05H: block is protected.

```

**Rename of block (44H)**

Change the block name.

```

entry
  (cl) = 06H
  (ch) = slot number o/d
  x = address of block name character string
  y = address of new block name

return
  x = last of old block name + 1
  y = last of new block name + 1
  cy = 0: no error
    1: error
    a = 00H: slot (card) is protected.
            01H: no specified slot.
            04H: block is not found.

            05H: block is protected.
            09H: the same block name exists.

```

### Transfer of block (43H)

Transfer the block to the address from the specified address by the specified size.

entry

(cx) = 0006H

i = 43H

x = address from which transferred

y = address to which transferred (*destination*)

(si) = size to be transferred

return:

x = completion address from which transferred + 1

y = completion address to which transferred + 1

### Creation of memory block (45H)

Create new memory block.

entry

(cl) = 06H

(ch) = slot number

i = 45H

x = address of block name

return

cy = 0: no error

y = lead address of created block

x = last of block name + 1

cy = 1: error

a = 00H: slot (card) is protected.

01H: no specified slot.

09H: the same block name exists.

0CH: insufficient memory

### Deletion of memory block (46H)

Delete the memory block.

```
entry
    (cl) = 06H
    (ch) = slot number
        i = 46H
        x = address of block name

return
    cy = 0: no error
    cy = 1: error
        a = 00H: slot (card) is protected.
        01H: no specified slot.
        02H: block is not found.
        05H: block is protected.
```

### Creation of memory block 2 (48H)

Create the memory block at the lead of the blocks.

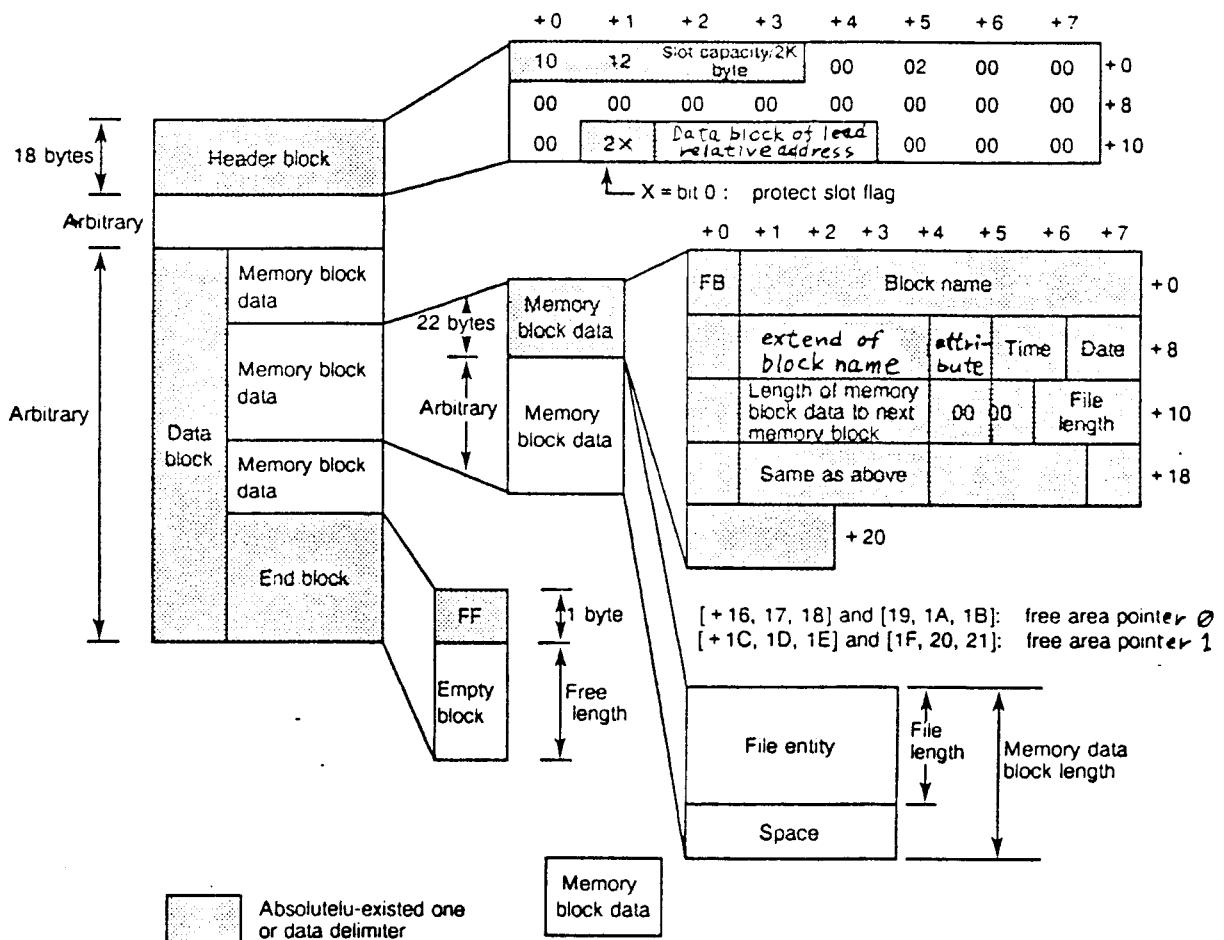
```
entry
    (cl) = 06H
    (ch) = slot number
        i = 48H
        x = lead address of block name
        y = size of memory block you want to create

return
    cy = 0: no error
        x = last of block name + 1
        y = lead address of block created
    cy = 1: error
        a = 00H: slot (card) is protected.
        01H: no specified slot.
        09H: the same block name exists.
        0CH: insufficient memory
```

### Parameter of memory block device

- |                   |                                   |
|-------------------|-----------------------------------|
| (BFC09H, 3 bytes) | lead address of slot 2 (S3:)      |
| (BFC0CH, 2 bytes) | capacity of slot 2/2K bytes (S3:) |
| (BFC0FH, 3 bytes) | lead address of slot 1 (S2:)      |
| (BFC12H, 2 bytes) | capacity of slot 1/2K bytes (S2:) |
| (BFC15H, 3 bytes) | lead address of slot 0 (S1:)      |
| (BFC18H, 2 bytes) | capacity of slot 0/2K bytes (S1:) |
| (BFCDEH, 3 bytes) | last address of slot 0 (S1:) + 1  |

### Structure of slot



## No. 07 2.5" FDD Driver (X: Y:)

This driver cannot be used.

### Command List

- Standard character device command  
 $08_{H}$ - $0F_{H}$  error
- Standard block device command  
 $10_{H}$ - $1F_{H}$  error
- Special device command  
 $20_{H}$ - $2F_{H}$  error
- Command peculiar to each device  
 $3F_{H}$  error  
 $40_{H}$  initialization of each parameter

### Initialization of each parameter ( $40_{H}$ )

entry  
(cx) =  $0007_{H}$   
i =  $40_{H}$   
a = level of initialization  
0: all reset  
1: reset  
2: off  
3: on

## No. 08 System Control Driver

This driver cannot be used.

### Command List

- Standard character device command  
 $08_{\text{H}}-0F_{\text{H}}$  error
- Standard block device command  
 $10_{\text{H}}-1F_{\text{H}}$  error
- Special device command  
 $20_{\text{H}}-2F_{\text{H}}$  error
- Command peculiar to each device
  - $3F_{\text{H}}$  error
  - $40_{\text{H}}$  no processing
  - $41_{\text{H}}$  power OFF
  - $42_{\text{H}}$  secure of work
  - $43_{\text{H}}$  execution of BASIC
  - $44_{\text{H}}$  (polynomial evaluation)
  - $45_{\text{H}}, 46_{\text{H}}$  Obtain processing address from the intermediate code.

### Power off ( $41_{\text{H}}$ )

Stop all of interrupt, switch off the LCD and stop the CPU.  
With pressing ON key, original state returns from this routine.

entry                   $\leftarrow \cdot L$   
 $(\text{cx}) = 0008_{\text{H}}$   
 $i = 41_{\text{H}}$

return  
nil

## Secure the work (42H)

Specify one of 21 work area pointers on the system memory and change the area size.  
Work area itself is secured by dividing slot 0 (S1:).  
Condensation of slot of memory area may be necessary before executing this comm

### entry

(cx) = 0008H  
i = 42H  
x = work area pointer  
    BFCDEH: for U stack  
    BFCE1H: for S stack  
    BFCE4H: reserve  
    BFCE7H: reserve  
    BFCEAH: reserve  
    BFCEDH: reserve  
    BFCF0H: reserve  
    BFCF3H: reserve  
    BFCF6H: reserve  
    BFCF9H: reserve  
    BFCFCH: reserve  
    BFCFFH: reserve  
    BFD02H: reserve  
    BFD05H: reserve  
    BFD08H: reserve  
    BFD0BH: reserve  
    BFD0EH: BASIC work  
    BFD11H: reserve  
    BFD14H: reserve  
    BFD17H: IOCS work  
    BFD1AH: machine language area  
y = size you want to secure

### return

cy = 0: no error  
cy = 1: error: insufficient memory

## Execution of BASIC (43H)

Execute the specified intermediate language character string.

entry

(cx) = 0008H  
i = 43H  
x = intermediate language character string address  
for BASIC  
✓  
a:  
bit 1 = 0: TROFF  
1: TRON  
bit 2 = 0: program execution  
1: manual execution  
bit 4 = 0: PRO mode  
1: RUN mode  
bit 0 = 0: normal operation  
1: step operation  
(execution)

return

x = data up to position where executed  
cy = 0: no error  
1: error  
a = error code of BASIC

## Acquire the address for processing from the intermediate code (45H, 46H)

entry

(cx) = 0008H  
i = 45H: When command address is needed.  
46H: When function address is needed.  
a = intermediate code of BASIC

return

cy = 0: no error  
y = address for processing routine (command address)  
cy = 1: error  
y = address of syntax error routine

## No. 09 Function Driver

Device unable to use.

### Command List

- Standard character device command  
 $08_{H}-0F_{H}$  error
- Standard block device command  
 $10_{H}-1F_{H}$  error
- Special device command  
 $20_{H}-28_{H}$  error

entry

(cy) = A

(ch) = B

i = C

when adding D and E,

[BFE03H] = D

[BFE04H] = E /

when X is necessary,

(bp)-(bp + 14) = X

also when Y is necessary,

(bp + 14)-(bp + 29) = Y

return

cy = 0: no error

(bp + 0~15) = pointer to number or character string

2-variable function, bp → bp + 15 in comparison

cy = 1: error

2-variable function, bp → bp + 30 in comparison

1-variable function, bp → bp + 15 in comparison

Individual command of each device

Function		A	B	C		
Numerical value function	2-variable function	Addition $Y + X \rightarrow X$	9	0	47H	
		Subtraction $Y - X \rightarrow X$			48H	
		Multiplication $Y * X \rightarrow X$			49H	
		Division $Y / X \rightarrow X$			4AH	
		Power $Y^X \rightarrow X$			4BH	
	1-variable function	EXP $e^X \rightarrow X$			4CH	
		SIN $\sin X \rightarrow X$			4DH	
		COS $\cos X \rightarrow X$			4EH	
		TAN $\tan X \rightarrow X$			4FH	
		SIN <sup>-1</sup> $\sin^{-1} X \rightarrow X$			50H	
		COS <sup>-1</sup> $\cos^{-1} X \rightarrow X$			51H	
		TAN <sup>-1</sup> $\tan^{-1} X \rightarrow X$			52H	
		DEG $X \rightarrow \text{DEG} \rightarrow X$			53H	
		DMS $X \rightarrow \text{DMS} \rightarrow X$			54H	
		ABS $ X  \rightarrow X$			55H	
Comparison	Numerical value	INT $\text{int} X \rightarrow X$			56H	
		SGN $\text{sgn} X \rightarrow X$			57H	
		RND $\text{rnd} X \rightarrow X$			58H	
		SQR $\sqrt{X} \rightarrow X$			59H	
		LOR $\log X \rightarrow X$			5AH	
		LN $\ln X \rightarrow X$			5BH	
		$Y < > X$			41H	
		$Y < X$			42H	
		$Y > X$			43H	
		$Y = X$			44H	
		$Y \leq X$			45H	
		$Y \geq X$			46H	

Function		A	B	C			
Comparison	Character	9	0	Y<>X	70H		
				Y<X	71H		
				Y>X	72H		
				Y=X	73H		
				Y≤X	74H		
				Y≥X	75H		
Conversion	Decimal → Binary conversion			7EH			
	Binary → Decimal conversion			7FH			
Character string operational function	ASC			76H			
	CJR \$ C H R \$			77H			
	STR \$			78H			
	VAL			79H			
Matrix operation	Addition	9		41H			
	Subtraction			42H			
	Multiplication			43H			
	Division			44H			
	inverse			45H			
	Addition to scholar		1	46H	CALL after entering scholar value in X.		
	Subtraction from scholar			47H			
	Multiply scholar			48H			
	Multiply X-1 scholar			49H			
	Replace X with Y			4AH			
	Transposed matrix			4BH			
	Value of determinant			4CH	Answer enters in x.		
	Reversion of symbol			4DH			
	Square			4EH			
	Store X in M			4FH			
	Call M to X			50H			
	Add M and X			51H			

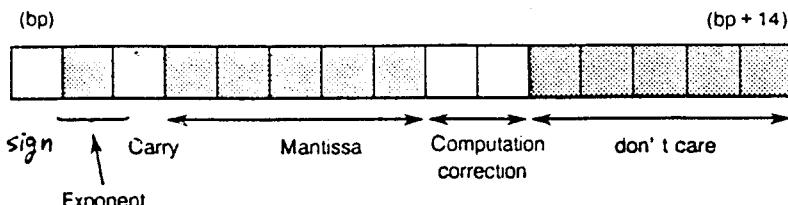
Function		A	B	C		
Matrix operation	Put X to MA - MZ	9	1	52H	A-Z	
	Put MA - MZ to X			53H	A-Z	
	Simultaneous equations			54H		
	Balance of simultaneous equations			55H		
Statistics regression	1-variable statistics	9	2	41H	0-8 X sequence	255
	Line regression			42H		
	Exponential regression*			43H		
	Logarithm regression*			44H		
	Power regression*			45H		
	Reciprocal regression*			46H		
	Secondary regression*			46H		
	Third regression*			47H		

## Type of Variable

### Internal format of numeric value (at execution of operation)

- Single accuracy numeric value

Single accuracy numeric value consists of 15 bytes. It is possible to represent numbers from  $\pm 1 \times 10^{-99}$  to  $\pm 9.99999999999999 \times 10^{-99}$  and 0 with combination of exponent, mantissa sign and mantissa.



~~sign~~

~~Zero~~ / Eight is used when the mantissa is positive.  
~~ZERO~~ Eight is used when the mantissa is negative.

#### Exponent

The exponent is expressed using hexadecimal. The negative numbers are expressed using a complement.

9D (10<sup>-99</sup>)-63 (10<sup>-99</sup>)

#### Carry

This is used only during operation. Normally, it is reset to 0.

#### Mantissa

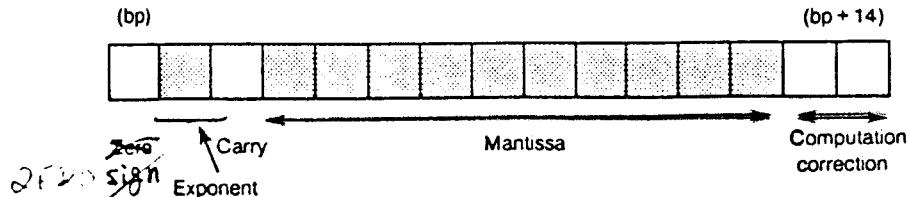
10-digit mantissa of numerical value is stored in memory with BCD code.

#### Computation correction

Computation correction is performed only during computation. Normally, it is reset to 0 after rounding off.

- Double accuracy numeric value

Double accuracy numeric value consists of 15 bytes and is able to express numbers from  $\pm 1 \times 10^{-99}$  to  $\pm 9.99999999999999 \times 10^{99}$  and 0.



### Mantissa sign

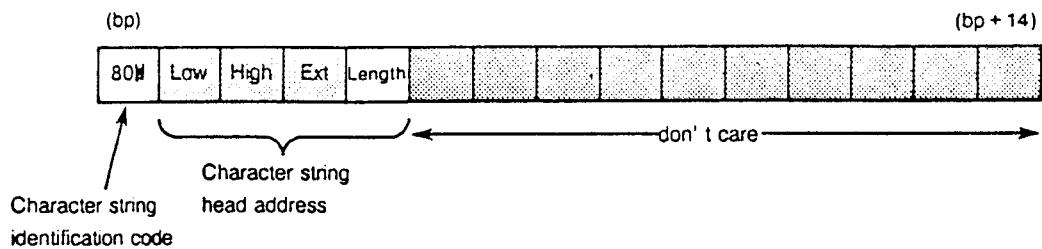
- 1 is used when the mantissa is positive.
- 9 is used when the mantissa is negative.

### Mantissa

20-digit mantissa of numeric value is stored in memory with BCD code.  
Other portions are same as single accuracy numeric value.  
The above expression method is expression at operation. In the text or variable, it is expressed in the style excepting carry, computation correction and don't care.

## Internal expression of character string

When a character string is stored in the variable, it is stored with ASCII code. At operation of character string etc., when the processing is carried out in the CPU, internal expression is composed of 15 bytes (effective data 5 bytes) as a character string information.



## IOCS Special Technique

Area from address written in  $BFD17_H$  to address written in  $BFD1A_H$  is assigned, as the working area of IOCS. Address written in  $BFD17_H$  is also written in  $E6_H$  of internal RAM. This area is generally  $246_H$  bytes.

Some buffers are included in this area. Following three buffers will become effective by changing the size.

~~Keyboard buffer: It is possible to change to more than 32 characters.~~

SIO buffer: Buffer of serial interface can be expanded.

FCB area: Maximum open-capable number can be extended to more than 5.

Pointer informations are shown in the figure.

Example: Refer the following program. Set the maximum opencapable file number at 10.

Execute as: (1) Input program

(2) RUN

(3) When the menu appears, select BASIC mode and input GOTO\*.

The following table shows meaning of some parameters.

```

10 P=&BFD17:GOSUB *P:A1=Q
20 P=&BFD1A:GOSUB *P:A2=Q
30 W=(10-PEEK (A1+&27))*&1F
40 S=(A2-A1)+W:S1=S/&100
50 POKE &BFE03,&17,&FD,&B,S-INT S1*&100,S1 AND &FF,S1/&100:CALL &FFFFD8
60 END
70 *P:Q=PEEK P+PEEK (P+1)*&100+PEEK (P+2)*&10000:RETURN
80 *W:Q=PEEK P+PEEK (P+1)*&100:RETURN
90 *
100 P=&BFD17:GOSUB *P:A1=Q
110 W=(10-PEEK (A1+&27))*&1F
? 120 POKE A1+&27,10
130 P=A1+&28:GOSUB *W:Q=W+Q
140 POKE P,Q AND &FF,Q/&100
150 P=A1+&2B:GOSUB *W:Q=W+Q
160 POKE P,Q AND &FF,Q/&100
170 P=&FFFFD:GOSUB *P
180 CALL Q

```

## Pointer information

P is set as (internal RAM E6H + 0, 1, 2) or [BFD17H + 0, 1, 2]

address	size
P + Ø	each parameter 5AH
P + [P + 10, 11H]	keyboard buffer [P + 14H] buffer size [P + 14H]
P + [P + 16, 17H]	Amount of buffer when [P + 1EH, 1FH] Xon. SIO buffer [P + 1CH, 1DH] buffer size [P + 1CH, 1DH]
P + [P + 25, 26H]	FCB area [P + 27H] maximum OPEN capable file number [P + 27H]*1FH
P + [P + 28, 29H]	Other buffer
P + [P + 2BH, 2CH] [BFD1AH + 0, 1, 2]	Other work

address	Meaning
[P + 8H]	LCD display start offset value Set in LCD in every approx. 5 min.
[P + 9H]	Information of cursor form bit 0-2 = 0 underline = 1 double underline = 2 full mark = 3 full space = 4 insert mark bit 3 = 0 no blink = 1 with blink But, when all bits are 0, cursor is not displayed.
[P + BH, CH]	Relative position of cursor screen buffer
[P + EH, FH]	During power is ON, counter to effect increment in approx. 5 sec. interval.

## BASIC

### Internal expression of BASIC program

BASIC of this pocket computer, same as other microcomputer, uses the intermediate language to store BASIC program. By converting into the intermediate language, memory saving and operating speed have been improved.

Following table shows the command and intermediate code of function. Actual program is converted as follows.

Data = (hexadecimal)	Description	
0D		Code showing head of BASIC program
00		
0A	10	Expression of line 10 in decimal
04		Line length (4 bytes by inclusion of stop code)
FE		
61	INPUT	Intermediate code of INPUT
41	A	Character code of A
0D		Delimiter of 1 line
00		
14	20	Expression of line 20 in decimal
04		Line length (4 bytes by inclusion of stop code)
FE		
60	PRINT	Intermediate code of PRINT
41	A	Character code of A
0D		Delimiter of 1 line
00		
1E	30	Expression of line 30 in decimal
06		Line length (6 bytes by inclusion of stop code)
FE		
2B	GOTO	Intermediate code of GOTO
1F		Line number discrimination code
00		
0A	10	Expression of line 10 in decimal
0D		Delimiter of 1 line
FF		Code showing termination of BASIC program

Line number discrimination code ( $1F_H$ ) is put in front of line number to which GOTO command in the program (statement) jumps. Actual line number is stored in memory by 2-byte binary code. The 2 bytes are stored in the order from the upper rank to lower rank contrary to normal case.

In addition, skip number discrimination code ( $1E_H$ ) and real number discrimination code ( $1D_H$ ) are provided.

#### Skip number discrimination code, real number discrimination code

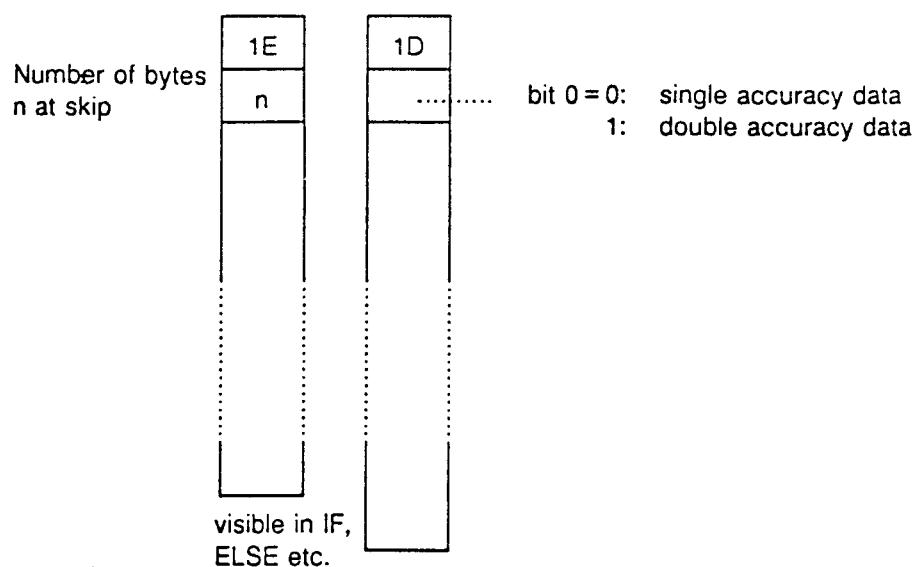


Table for BASIC intermediate code

00:(RESERVED)	10:RUN	20:CSAVE	30:DIM	40:LTEXT
01:	11:NEW	21:OPEN	31:CALL	41:GRAPH
02:	12:CONT	22:CLOSE	32:POKE	42:LF
03:	13:PASS	23:SAVE	33:GPRINT	43:CSIZE
04:	14:LIST	24:CONSOLE	34:PSET	44:COLOR
05:	15:LLIST	25:RANDOMIZE	35:PRESET	45:
06:	16:CLOAD	26:DEGREE	36:BASIC	46:DEFDBL
07:	17:MERGE	27:RADIAN	37:TEXT	47:DEFSNG
08:	18:LOAD	28:GRAD	38:	48:
09:	19:RENUM	29:BEEP	39:	49:
0A:	1A:AUTO	2A:WAIT	3A:ERASE	4A:
0B:BTEXT\$	1B:DELETE	2B:GOTO	3B:LFILES	4B:
0C:BDATA\$	1C:FILES	2C:TRON	3C:KILL	4C:
0D:MEM\$	1D:INIT	2D:TROFF	3D:COPY	4D:
0E:	1E:	2E:CLEAR	3E:NAME	4E:
0F:	1F:	2F:USING	3F:SET	4F:
50:CLS	60:PRINT	70:PAINT	80:MDF	90:FACT
51:LOCATE	61:INPUT	71:OUTPUT	81:REC	91:LN
52:TO	62:GOSUB	72:APPEND	82:POL	92:LOG
53:STEP	63:	73:AS	83:ROT	93:EXP
54:THEN	64:LPRINT	74:ARUN	84:DEC1	94:SQR
55:ON	65:RETURN	75:AUTOGOTO	85:HEX	95:SIN
56:IF	66:RESTORE	76:ELSE	86:TEN	96:COS
57:FOR	67:CHAIN	77:RESUME	87:RCP	97:TAN
58:LET	68:GCURSOR	78:ERROR	88:SQU	98:INT
59:REM	69:LINE	79:KEY	89:CUR	99:ABS
5A:END	6A:LLINE	7A:	8A:HSN	9A:SGN
5B:NEXT	6B:RLINE	7B:	8B:HCS	9B:DEG
5C:STOP	6C:GLCURSOR	7C:	8C:HTN	9C:DMS
5D:READ	6D:SORGN	7D:	8D:ASN	9D:ACS
5E:DATA	6E:CRotate	7E:	8E:AHC	9E:AC\$
5F:PAUSE	6F:CIRCLE	7F:	8F:AHT	9F:ATN
A0:RND	B0:EOF	C0:ERN	D0:ASC	E0:
A1:AND	B1:DSKF	C1:ERL	D1:VAL	E1:
A2:OR	B2:LOF	C2:	D2:LEN	E2:
A3:NOT	B3:LOC	C3:	D3:	E3:
A4:PEEK	B4:	C4:	D4:	E4:
A5:XOR	B5:	C5:	D5:	E5:
A6:	B6:NCR	C6:	D6:	E6:
A7:EVAL	B7:NPR	C7:	D7:	E7:
A8:	B8:	C8:	D8:	E8:
A9:	B9:	C9:	D9:	E9:INKEY\$
AA:	BA:	CA:	DA:	EA:MO\$
AB:	BB:	CB:	DB:	EB:LEFT\$
AC:	BC:	CC:	DC:	EC:RIGHT\$
AD:POINT	BD:	CD:	DD:	ED:
AE:PI	BE:AER	CE:	DE:	EE:
AF:FRE	BF:CUB	CF:	DF:	EF:

Intermediate code is expressed in 2 bytes for FEH plus value in the above table.  
 For example, INPUT command is stored in memory in the order of FEH, 01H.

### Location to store data

Data that may be necessary when operating BASIC are roughly classified as follows.

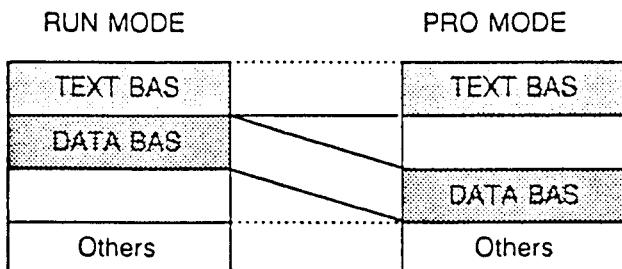
- ◎ Text data (program)
- ◎ Variable data
- ◎ Function key data
- ◎ AER data

All of these data exist as the block of slot file. Set as FILES "S1:" and press the [RET] key.

If no alteration is in each, data is stored in the following file name.

TEXT . BAS  
DATA . BAS  
FUNCKEY .  
AER .

Explanation on text data and variable data. Below is memory map in PRO mode and RUN mode.



Point to pay attention is that free space of memory moves according to RUN mode and PRO mode. Structure of the block file requests to separate the time when variable is increased and the time when text (program) is increased.

## Interrupt

Interrupt processing for PC-E500 is described below concretely.

- **Interrupt service**

Eight kinds of interrupt are provided in our pocket computer, but only one address for interrupt processing by CPU specification is on the ROM.

This interrupt is index service routine that jumps to each processing classified by the factor.

It is advantageous to the users that this interrupt service sub-routine has 8 addresses for processing separated by its factor on the RAM. Accordingly, we can rewrite freely the addresses for processing into own interrupt processing program.

### List of interrupt factor

(1)	Fast timer interrupt	All of key scan including break key
(2)	Slow timer interrupt	Blinking of cursor, etc.
(3)	Key interrupt	not used
(4)	ON key interrupt	not used
(5)	SIO transmission interrupt	not used
(6)	SIO reception interrupt	completion of SIO reception
(7)	External interrupt	low battery
(8)	Software interrupt	not used

- **Description of each interrupt**

- **Fast timer interrupt**

Interrupt occurs in every constant time. Possible to select one interval of time arbitrarily from 4 msec and 16 msec. Ordinarily time interval is set in 16 msec.

(Internal RAM FD<sub>H</sub>) bit 1 = 0: 4 msec.  
= 1: 16 msec.

However, as this interrupt is made with the CPU clock divided, it is ignored when the CPU clock is stopped, that is, during execution of half or off command.

#### ○ Slow timer interrupt

An interrupt occurs in every constant time. Possible to select one interval of time arbitrarily from approx. 0.5 sec and approx. 2 sec. Usually, it is set in approx. 5 sec.

(Internal RAM FD<sub>H</sub>) bit 2 = 0: approx. 0.5 sec  
= 1: approx. 2 sec

However, as this interrupt is made with the sub-clock divided, interrupt is ignored when sub-clock is stopped, that is, during execution of off command.

Also, time interval fluctuates considerably in compliance with power voltage.

#### ○ Key interrupt

In the key matrix, when any of key-input ports (K10-K17) is level 1, that is, when (internal RAM F2H is not 0, an interrupt occurs.

### ○ On key interrupt

An interrupt occurs when ON key is pressed.

© SIO transmission interrupt

An interrupt occurs when the SIO has completed to transmit 1 byte.

#### ◎ SIO reception interrupt

An interrupt occurs when receiving 1 byte from the SIO.

### External interrupt

Request of interrupt from outside of the CPU. This is connected with battery checker in the pocket computer.

### ○ Software interrupt

If command ir is carried out, this interrupt occurs.

- Caution when creating interrupt processing program

For interrupt program

- {1} Complete processing is executed with "retf".
  - {2} Do not use U stack or S stack frequently. (Limited to several ten bytes)
  - {3} Display  
In case of continuous input in horizontal direction just before interrupt, if display output is executed during interrupt processing, irregular shape may appear on the display.
  - {4} Avoid overlapping of the program with the program just before a work area interrupts.  
Do not run several work.  
It is not permitted to operate same area of program just before interrupt except when receiving data.
  - {5} Keep aside the contents of the internal RAM. Return it to the original value after use.

(In case the program of just before interrupt is using BP as stack pointer, it is possible to use as work corresponding to BP.)

- Register related to interrupt

The following three registers are provided for interrupt.

- Interrupt status register isr (0FC<sub>H</sub>)

bit 0 = 16 msec timer  
 bit 1 = 0.5 sec timer  
 bit 2 = key  
 bit 3 = ON key  
 bit 4 = transmission  
 bit 5 = reception  
 bit 6 = low battery  
 bit 7 = 0

Interrupt is requested by 1. Completion of service makes 0.

- Interrupt enable register imr (0FB<sub>H</sub>)

bit 0 = 16 msec timer  
 bit 1 = 0.5 sec timer  
 bit 2 = key  
 bit 3 = ON key  
 bit 4 = transmission  
 bit 5 = reception  
 bit 6 = low battery  
 bit 7 = carry out all mask of interrupt.

Interrupt is permitted by 1. Interrupt is prohibited by 0.

- Interrupt during service register iisr (0EB<sub>H</sub>)

bit 0 = 16 msec timer  
 bit 1 = 0.5 sec timer  
 bit 2 = key  
 bit 3 = ON key  
 bit 4 = transmission  
 bit 5 = reception  
 bit 6 = low battery  
 bit 7 = execution of 'ir' command 0FE<sub>H</sub>

Bit = 1 while interrupt routine is executed.

Isr register is set in accordance with interrupt status. By setting imr, interrupt processing is controlled actually by the factor.

While executing interrupt processing routine, bit corresponding to iisr has become 1. And, bit of isr is reset.

If an interrupt occurred, the 'call f' control is carried out to address shown by vector according to the kind.

### Interrupt vector

All vector has 3 bytes.

- BFCC6<sub>H</sub>: fast timer interrupt
- BFCC9<sub>H</sub>: slow timer interrupt
- BFCCC<sub>H</sub>: key interrupt
- BFCCF<sub>H</sub>: ON key interrupt
- BFCD2<sub>H</sub>: SIO transmission interrupt
- BFCD5<sub>H</sub>: SIO reception interrupt
- BFCD8<sub>H</sub>: Outside interrupt
- BFCDB<sub>H</sub>: Software interrupt

### Relation among isr, imr, and iisr operations

