TECHNICAL REFERENCE MANUAL

PC-E500

SHARP CORPORATION 1990.3

INFORMATION SYSTEMS GROUP
PERSONAL EQUIPMENT DIVISION

Contents

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 $0FFFE4_H$. 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.

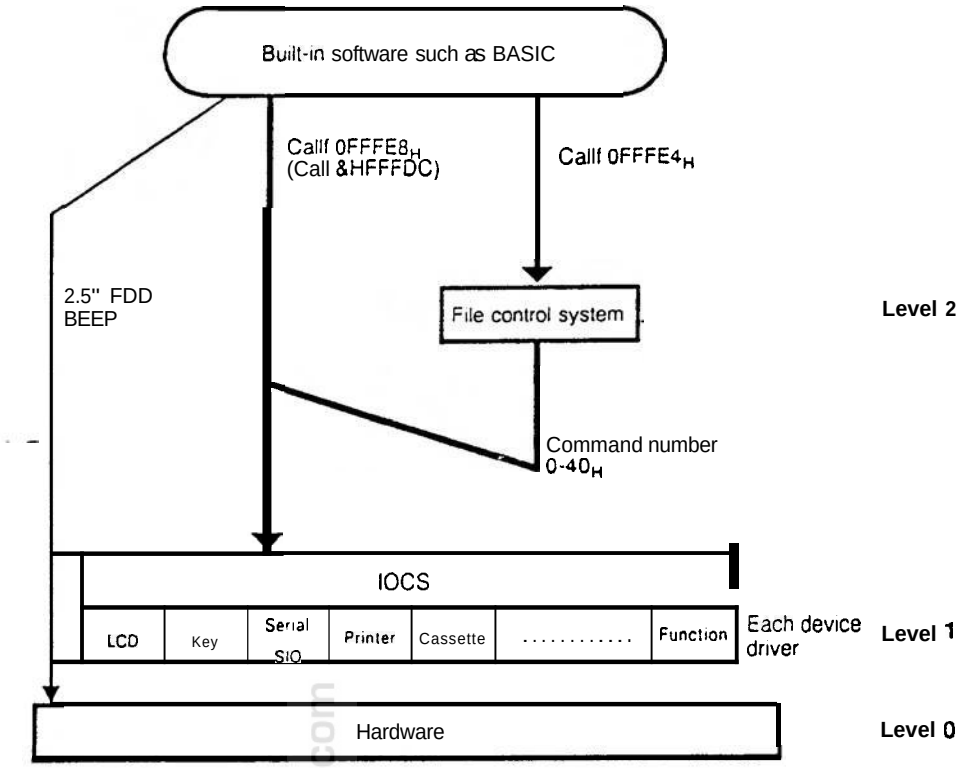
- Level 2: Level on which each device can be handled as a file. On this level, you can only open a file, receive the data and close the file.
- Level 1: Function of each device driver is utilized to its utmost on this level. Various functions can be used in accordance with device. You can handle a function 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.

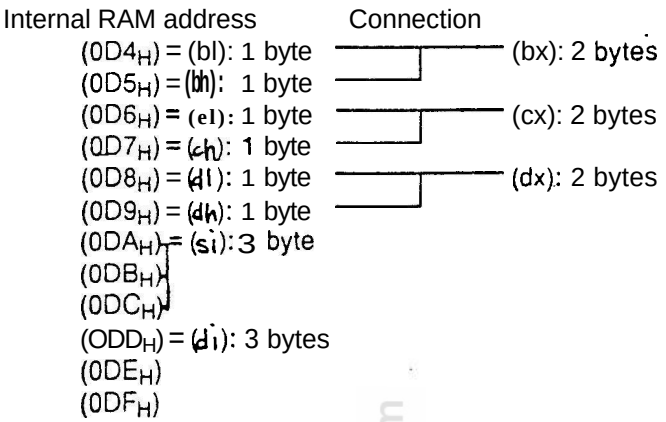
CHAPTER 2: Outline of IOCS (level 1)
Instruction of structure/extension of device driver
Explanation of call from file control system of device driver

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

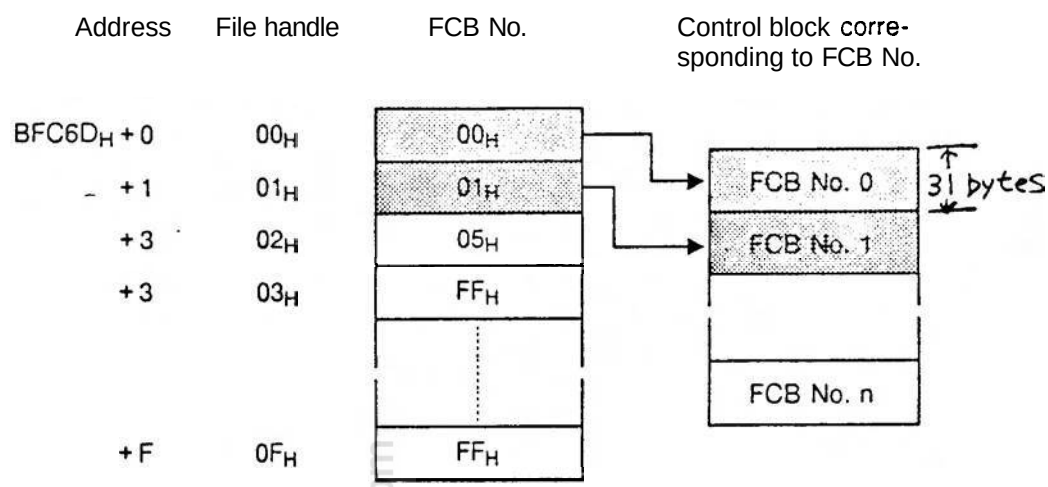
The contents of each function are shown below.

Function No.	
00_H	Create a file
01_H	Open a file
02_H	Close a file
03_H	Read a file block
04_H	Write a file block
05_H	Read a byte of file
06_H	Write a byte of file
07_H	Verify a file
08_H	Nondestructive read of file
09_H	Move a file pointer
$0A_H$	Read of various information on file
$0B_H$	Alteration of directory information of drive
$0C_H$	Search of corresponding file name
$0D_H$	Rename a file
$0E_H$	Delete a file
$0F_H$	Read of empty capacity of drive
10_H	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 00_H to $0F_H$ and $0FF_H$ are included in the FCB number.

00_H-$0F_H$: FCB number
FF_H: 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)	stdo:
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.

- 00_H **An** error occurred in the device and aborted.
- 01_H The parameter is beyond the range.
- 02_H The specified file does not exist.
- 03_H The specified pass code does not exist.
- 04_H The number of files to be opened exceeds the limit.
- 05_H The file whose processing is not permitted.
- 06_H Ineffective file handle was attempted.
- 07_H Processing is not specified by open statement.
- 08_H The file is during open.
- 09_H The file name is duplicated.
- $0A_H$ The specified drive does not exist.
- $0B_H$ Error in data verification.
- $0C_H$ Processing of byte number has not been completed.
- FE_H Fatal low battery.
- FF_H 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 $FFFE4_H$. Explanations of each parameter are as follows.

Creating a file (00_H)

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 000000_H. 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 = 00_H
a = file attribute
bit 0 write protect
bit 1 invisible
x = the lead address of file name character string (00_H in last with shift JIS)

return

cf = 0
(cl) = File handle
a = File attribute
cf = 1
Error (a = 00_H, 03_H, 04_H, 05_H, 08_H, $0A_H$, FF_H)

Opening a file (01_H)

Open a file of the specified file name. File pointer is set at 00000_H. 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 = 01_H
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 (00_H in last with shift JIS)

return

cf = 0
(cl) = File handle
a = File attribute
cf = 1
Error (a = 00_H, 01_H, 02_H, 03_H, 04_H, 05_H, 08_H, $0A_H$, FF_H)

Closing a **file** (02_H)

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 = 02_H
(cl) = File handle

return

cf = 0
No error
cf = 1
Error (a = 00_H, 06_H, FF_H)

Reading a block of the file (03_H)

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 $1A_H$ (end of file) which specifies the whole file or by specifying the number of bytes to be read.

entry

i = 03_H
(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 $1A_H$ code.
(Pointer stops indicating $1A_H$. $1A_H$ 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 = 00_H, 06_H, 07_H, $0C_H$, FF_H)
x = Sex: address that read correctly.
y = Number of bytes that was read correctly.

Writing a block of the file (04_H)

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 = 04_H
(cl) = File handle
x = The lead address of data
y = Number of bytes to be written

return

cf = 0
x = Next data of the written data
y = Number of written bytes
cf = 1
Error (a = 00_H, 06_H, 07_H, $0C_H$, FF_H)
x = Next address that was written correctly.
y = Number of bytes that was written correctly.

Reading a byte of the file (05_H)

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

entry

i = 05_H
(cl) = File handle
a: bit 0 = 0 Fiie end is $1A_H$ dode.
(Pointer stops indicating $1A_H$. $1A_H$ is read.)
1 File end is a physical end of the file.
(Pointer stops indicating final byte of pointer + 1)

return

cf = 0
a = Data
b = Number of read data
cf = 1
Error (a = 00_H, 06_H, 07_H, FF_H)

Writing a byte of the file (06_H)

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

entry

i = 06_H
(cl) = File handle
a = Data

return

cf = 0
b = Number of read data
cf = 1
Error (a = 00_H, 06_H, 07_H, FF_H)

Verifying a file (07_H)

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 $1A_H$ code or in physical file end.

entry

i = 07_H
(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 $1A_H$ code.
(Pointer stops indicating $1A_H$. $1A_H$ 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 = 00_H, 06_H, 07_H, $0C_H$, FF_H)
x = Address of data that an error was occurred
y = Number of bytes verified

Nondestrucfive reading a file (08_H)

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

entry

i = 08_H
(cl) = File handle
a: bit 0 = 0 File end is $1A_H$ code.
(Pointer stops indicating $1A_H$. $1A_H$ 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 = 00_H, 06_H, 07_H, FF_H)

Moving a file pointer (09_H)

Move the specified amount of the file pointer with the specified method. Value from 00000_H to $FFFFF_H$ 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 = 09_H
(cl) = File handle
(si) = Number of bytes to move (3 bytes)
a = 00_H: Relative value from the file top **(24** bits without sign)
01_H: Relative value from present position **(24** bits with sign)
02_H: Relative value from file end + 1 (24 bits without sign)

return

cf = 0
(si) = File pointer value (3 bytes)
cf = 1
error (a = 00_H, 01_H, 06_H, 07_H, FF_H)

Reading various information of a file ($0A_H$)

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

entry

$i = 0A_H$
(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 + 0_H =	drive name + " : "	(6 bytes)
+ 6_H =	file name	(8 bytes)
+ E_H =	" . "	(1 byte)
+ F_H =	extension	(3 bytes)
+ 12_H =	attribute	(1 byte)

cf = 1
error (a = 00_H, 06_H, FF_H)

Changing directory information of **drive** ($0B_H$)

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

entry

$i = 0B_H$
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 ~~(00_H 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 00_H when not specified)
+ 3 date information (all 00_H when not specified)
+ 5 file size (read only)
cf = 1
error (a = 00_H, 01_H, 02_H, 03_H, 08_H, $0A_H$, FF_H)

Searching for corresponding file name ($0C_H$)

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 = $0C_H$
a = 00_H Search for the back of the specified directory number
01_H Search for the front of the specified directory number
80_H Perform 00_H disregarding to be invisible
81_H Perform 01_H 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_H 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_H, 03_H, 09_H, $0A_H$, FF_H)

Renaming a file ($0D_H$)

Rename the file to the specified name.

entry

i = $0D_H$
x = Lead address of file name character string before renaming ~~(00_H in last with shift JIS)~~
y = Lead address of file name character string after renaming ~~(00_H in last with shift JIS)~~

return

cf = 0
none
cf = 1
error (a = 00_H, 02_H, 03_H, 05_H, 08_H, 09_H, $0A_H$, FF_H)

Deleting a file ($0E_H$)

Delete the specified file.

entry

$i = 0E_H$
x = Lead address of the file name character string to be deleted
~~(00_H in last with shift JIS)~~

return

cf = 0
nil
cf = 1
error (a = 00_H, 02_H, 03_H, 05_H, $0A_H$, FF_H)

Reading a free capacity of drive($0F_H$)

Examine free capacity of the specified drive.

entry

$i = 0F_H$
a = O
x = Drive name character string ~~(00_H in last with shift JIS)~~

return

cf = 0
(si) = Free capacity of drive (3 bytes)
cf = 1
error (a = 00_H, 03_H, FF_H)

Initializing a file control system (10_H)

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

entry

$i = 10_H$
a = 00_H Operation of initialization of the work and "a = 01_H"
01_H Release all FCB, file handle, and file buffer. Open standard input, standard output, standard listing out put.
02_H 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 00_H
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 00_H 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.

You can use IOCS by performing ~~fcall~~ CALLF of $FFFE8_H$. Command is used to operate file-control and peculiar to device.

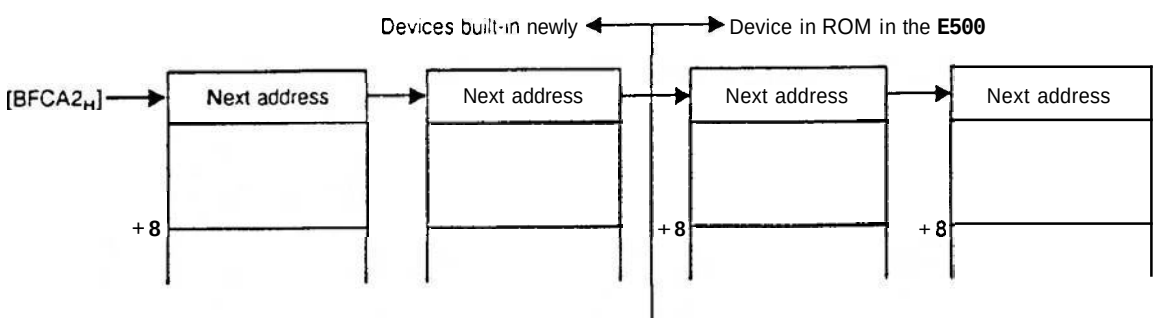
The IOCS consists of the IOCS main routine, IOCS routine of individual device, and IOCS header. As the IOCS is built in the system by the IOCS header, you can write a program assembled by the machine language and add the header to correspond with new device or to alter the function of existing device.

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. 00_H 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, $OFFFFF_H$ is set.

- ADevice number
  The number to represent this device. Device number is peculiarly assigned to each device.



In case of same device name, newly built-in device is selected.

- 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, $C7_H$ 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.

  $43_H, 4F_H, 4D_H, 31_H, 3A_H, 43_H, 4F_H, 4D_H, 32_H, 3A_H, 00_H$

  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, $A\,X_H$ 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	STDO:, 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	OOH
System	SYSTM:	8	0	OOH
Function	NIL	9	NIL	OOH

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.

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

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

- Command No. = 20_H-$3E_H$

  00_H An error occurred in the device and aborted.
  01_H The parameter is beyond the range.
  02_H The specified file does not exist.
  03_H The specified pass code does not exist.
  04_H The number of files to be opened exceeds the limit.
  05_H The file processing is not allowed.
  06_H Invalid file handle was used.
  07_H Processing is not specified when opened.
  08_H The file is during open.
  09_H The corresponding file does not exist.
  $0A_H$ The file name is duplicated.
  $0B_H$ The specified drive does not exist.
  $0C_H$ Verify error
  $0D_H$ Processing of the specified number of bytes has not been completed.
  FE_H Fatal low battery
  FF_H 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 (00_H-05_H)

Searching for the drive name (00_H)

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

entry

i = 00_H
x = drive name lead address

return

(cl) = device number
(ch) = drive number
cy = 0: no error
cy = 1: error (a = 01_H)

Checking the header address 1 (01_H)

Search the header address from the specified device number.

entry

i = 01_H
(cl) = device number

return

cy = 0: x = header address
cy = 1: error (a = 01_H)

Checking the Header Address 2 (02_H)

Search the header address from the specified device number.

entry

i = 02_H
(cl) = device number
x = the first header address to start searching for return

return

cy = 0: x = header address
cy = 1: error (a = 01_H)

Checking the drive name (03_H)

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

entry

i = 03_H
(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 20_H.
cy = 1: error (a = 01_H)

Processing the format (04_H)

Execute format of the device of the specified drive name.

entry

i = 04_H
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 = 01_H, same error code with BASIC)

All initialize (05_H)

Initialize all of the stored devices.

entry

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

return

nil

Processing the File of Standard Character Device (08_H-$0E_H$)

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 car, be handled as the standard character device.

Character device OPEN (08_H)

Execute initialization when the standard character device is opened.

entry
- i = 08_H
- (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 = 00_H, 02_H, 03_H, 08_H, 09_H, FE_H, FF_H)

Character device CLOSE (09_H)

Execute processing when the standard character device is closed.

entry
- i = 09_H
- (cl) = device number
- (ch) = drive number

return
- none

Read byte data ($0C_H$)

Read the data of 1 byte only.

entry
- i = $0C_H$
- (cl) = device number
- (ch) = drive number

return
- cy = 0: no error
  a = data
- cy = 1: error (a = 01_H, 02_H, 03_H, 04_H, 06_H, 09_H, FE_H, FF_H)

Write byte data ($0D_H$)

Write the data of 1 byte only.

entry
- i = $0D_H$
- (cl) = device number
- (ch) = drive number
- a = data

return
- cy = 0: no error
- cy = 1: error (a = 01_H, 02_H, 03_H, 04_H, 05_H, 08_H, FE_H, FF_H)

Read block data ($0A_H$)

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

entry
- i = $0A_H$
- (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 = 01_H, 02_H, 03_H, 04_H, 06_H, 09_H, FE_H, FF_H)

Write block 'data ($0B_H$)

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

entry
- i = $0B_H$
- (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 = 01_H, 02_H, 03_H, 04_H, 05_H, 08_H, FE_H, FF_H)

Nondestructive read **byte** data ($0E_H$)

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

entry
- i = $0E_H$
- (cl) = device number
- (ch) = drive number

return
- cy = 0: no error
  - a = data
- cy = 1: error (a = 01_H, 02_H, 03_H, 04_H, 06_H, 09_H, FE_H, FF_H)

File Processing of Standard Block Device (10_H-17_H)

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 = 10_H
- (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 = 01_H, 02_H, 03_H, FE_H, FF_H)

Media parameter block address (11_H)

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

entry

$i = 11_H$

(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) = $\log_2$ (number of directories entered in a sector)

(x + 5) = 00_H

(x + 6) = 01_H

(x + 7, 8) = number of spare sectors (2 bytes of low and hi) − lead sector of FAT

(X + 9) = 08_H

(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 bytes per sector/32

Number of sectors occupied by one FAT = [number of usable sectors + 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 directories/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(12_H)

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

entry

i = 12_H
(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 = 01_H, 02_H, 03_H, 04_H, 06_H, 09_H, FE_H, FF_H)
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 **(13_H)**

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

entry

i = 13_H
(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 = 00_H, 01_H, 02_H, 03_H, 04_H, 05_H, 08_H, FE_H, FF_H)
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 (14_H)

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

entry

i = 14_H
(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 = 00_H, 01_H, 02_H, 03_H, 04_H, 05_H, 07_H, 08_H, FE_H, FF_H)
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* (15_H)

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

entry

i = 15_H
(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 = 01_H, 02_H, 03_H, 04_H, 07_H, FE_H, FF_H)
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 (16_H)

The specified drive attribute is read.

entry
- i = 16_H
- (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 = 01_H, 02_H, 03_H, FE_H, FF_H)

Get sector address (17_H)

The address of the specified sector is got.

entry
- i = 17_H
- (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 = 01_H, 02_H, 03_H, FE_H, FF_H)

Processing of Special File Device (20_H-$2F_H$)

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_H	00_H	Creating a file
21_H	01_H	Opening a file
22_H	02_H	Closing a file
23_H	03_H	Reading a block of the file
24_H	04_H	Writing a block of the file
25_H	05_H	Reading a byte of the file
26_H	06_H	Writing a byte of the file
27_H	07_H	Verifying a file
28_H	08_H	Non-destructive reading a file
29_H	09_H	Moving a file pointer
$2A_H$	$0A_H$	Reading various information of the file
$2B_H$	$0B_H$	Changing a drive directory information
$2C_H$	$0C_H$	Searching for corresponding file name
$2D_H$	$0D_H$	Renaming a file
$2E_H$	$0E_H$	Deleting a file name
$2F_H$	$0F_H$	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

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.

CALL &HFFFE8
(CALLFØFFFE8H

And, command that requires only register il, (cl), (ch) is called after writing each value for address $0BFE00h_H$ to $0BFE02_H$ with setting as'follows.

CALL&HFFFDC

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

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

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_H ~~No special processing~~ Non - operation
09_H ~~No special processing~~ Non - operation
$0A_H$ Error
$0B_H$ Output to display
$0C_H$ Error
$0D_H$ Output to display
$0E_H$ Error
$0F_H$ Error

- Command for standard block device
  $1O_H$-$1F_H$ Error

- Command for special device
  20_H-$2F_H$ Error

- Command peculiar to each device
  $3F_H$ Formatting
  40_H Initialization of each parameter
  41_H 1 character output to arbitrary position
  42_H n character output to arbitrary position
  43_H Not used
  44_H Sets cursor position
  45_H Sets type of cursor display
  46_H Performs symbol display
  47_H n line scroll-up
  48_H n line scroll-down
  49_H Clears 1 line.
  $4A_H$ Displays 8-dot pattern.
  $4B_H$ Reads 8-dot patterns.
  $4C_H$ Displays 1 dot.
  $4D_H$ Reads 1 dot.
  $4E_H$ Displays line.
  $4F_H$ Paints out the box.
  50_H Sets display state.
  51_H Clears display.
  52_H Clears from the specified position.
  53_H (Deletes one line.)
  54_H Inserts n line.
  55_H Transfers one-line dot pattern to memory.
  56_H Displays dot pattern on the memory.
  57_H Displays regardless of display range.
  58_H Displays guide line.

Format ($3F_H$)

entry

(cx) = 0000_H
il = $3F_H$

return

nil

Initializing LCD driver (40_H)

Initialize work area of LCD driver.

entry

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

return

nil

One character output to arbitrary position (41_H)

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

entry

(cx) = 0000_H
i = 0041_H
(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 (44_H)

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

entry

(cx) = 0000_H
i = 0044_H
(bi) = cursor x coordinate
(bh) = cursor y coordinate

return

cy =An attempt was made to display outside range.

Character output to arbitrary position(42_H)

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

entry

(cx) = 0000_H
i = 0042_H
(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 (45_H)

Set the cursor type.

entry

(cx) = 0000_H
i = 0045_H
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 =

		2 1 0
0:	Underline	0 0 6
1:	Double underline	0 0 1
2:	Full mark	0 1 0
3:	Full space	0 1 1
4:	Insert mark	1 0 0

return

nil

Symbol display (46_H)

Execute ON/OFF of symbol.

entry

(cx) = 0000_H
i = 0046_H
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 (47_H)

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

entry

(cx) = 0000_H
i = 0047_H
(bx) = 0000_H
a =number of lines to be scrolled.

return

nil

n line scroll-down (48_H)

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

entry

(cx) = 0000_H
i = 0048_H
(bx) = 0000_H
a =number of lines to be scrolled.

return

nil

Clear line (49_H)

Clear the specified line.

entry

(cx) = 0000_H
i = 0049_H
(bh) = Y-coordinate of the line to be cleared.

return

nil

8-dot pattern display ($4A_H$)

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

entry

(cx) = 0000_H
i = $004A_H$
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.
0: light up
1: clear
2: reverse

@BFC96H
(work area)

return

nil

8-dot pattern read ($4B_H$)

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

entry

(cx) = 0000_H
i = $004B_H$
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 ($4C_H$)

Displayed arbitrary 1 dot is lit and cleared.

entry

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

return

nil

1-dot read ($4D_H$)

Displayed arbitrary 1 dot is read.

entry

(cx) = 0000_H
i = $004D_H$
x = X-coordinate of dot
y = Y-coordinate of dot

return

a = data (1 = light up)

Line display ($4E_H$)

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

entry

(cx) = 0000_H
i = $004E_H$
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)

return

0BFC2AH (work area)

[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 ($4F_H$)

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

entry

(cx) = 0000_H
i = $004F_H$
x = X-coordinate of diagonal
y = Y-coordinate of diagonal
(bx) = X-coordinate of diagonal
(dx) = Y-coordinate of diagonal
[dotsop] =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 (50_H)

ON and OFF of display

entry

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

return

nil

Clearing of display 1 (51_H)

Clear the display completely.

entry

(cx) = 0000_H
i = 0051_H

return

nil

Clearing of display 2 (52_H)

Clear the display from the specified position.

entry

(cx) = 0000_H
i = 0052_H
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 (54_H)

Insert n line.

entry

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

return

nil

Reading of one-line dot pattern (55_H)

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

entry

(cx) = 0000_H
i = 0055_H
(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 (56_H)

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

entry

(cx) = 0000_H
i = 0056_H
(bh) =Y-coordinate of line you want to write.
x =data address

return

nil

Displaying regardless of range (57_H)

Display one character regardless of the specified range of display.

entry

(cx) = 0000_H
i = 0057_H
(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 (58_H)

Character string is displayed with guide line [ ].

entry

(cx) = 0000_H
i = 0058_H
(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 FF_H.
A character string may be composed of any number of characters, but characters over 7 cannot be displayed.
When the FE_H code is on the lead of one character string, portion succeeding to FE_H to the last of that character string is displayed until next FE_H appears.
Also, when the FE_H code is at the location other than the lead of one character string, portion from the lead of char acter string to the from of it is displayed.

return

nil

Parameter work

lcd mode ($0BFCA1_H$, 1 byte)
Setting of display mode. Character output succeeding bit 6 is displayed reversely.

- lcd crsr x ($0BFC9B_H$, 1 byte)
  X-coordinate of cursor

- lcd crsr y ($0BFC9C_H$, 1 byte)
  Y-coordinate of cursor

- lcd width ($0BFC9D_H$, 1 byte)
  Number of digits possible to display is shown.

- lcd height ($0BFC9E_H$, 1 byte)
  Number of lines possible to display is shown.

- scrn crsr x ($0BFC27_H$, 1 byte)
  X-coordinate displayed next to **stdo :/scrn : device** is shown.

scrn srst y ($0BFC28_H$, 1 byte)
Y-coordinate displayed next to stdo: /scrn: device is shown.

- dotsop ($0BFC96_H$, 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 ($0BFC2A_H$, 2 bytes)
  Specify the dot pattern that is displayed by the line or boxfull with 16 dots.

- ($0BFC29_H$, 1 byte)
  Control code succeeding bit *6* is displayed.
  (No control operation is executed.)

Storing address of character font
($BFC87_H$, 3 bytes) Storing address for character font 00_H-$1F_H$
($BFC90_H$, 3 bytes) Storing address for character font 20_H-$7F_H$
($BFC8A_H$, 3 bytes) Storing address for character font 80_H-$9F_H$
($BFC93_H$, 3 bytes) Storing address for character font $A0_H$-CF_H
($BFC8D_H$, 3 bytes) Storing address for character font $E0_H$-$F0_H$
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_H ~~No processing~~ Non-operation
  09_H ~~No processing~~ Non-operation
  $0A_H$ Key input
  $0B_H$ Error
  $0C_H$ Key input
  $0D_H$ Error
  $0E_H$ Key input
  $0F_H$ Error

- Command for standard block device
  $1O_H$-$1F_H$ Error

- Command for special device
  20H-2FH Error

- Command peculiar to each device
  $3F_H$ Formatting
  40_H Initialization of each parameter
  41_H Read out matrix code
  42_H Non-destructive read out of matrix code
  43_H Key read out
  44_H Set data to key buffer
  45_H Buffer clear
  46_H Display of function key

Format ($3F_H$)

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

entry

(cx) = 0001_H
i = $3F_H$

return

nil

Pointer of conversion table (each 3 bytes)
[0BFC2D$_H$]: for 1-byte code
[0BFC30$_H$]: for 2-byte code
[0BFC33$_H$]: for 1-byte code of SHIFT
[0BFC36$_H$]: for 2-byte code of SHIFT
[0BFC39$_H$]: for 1-byte code of CTRL
[0BFC3C$_H$]: for 2-byte code of CTRL
62 bytes per one table

Initialization of key device driver (40$_H$)

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

return
nil

Read out of matrix code (41$_H$)

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 OFH is brought back.

entry
(cx) = 0001$_H$
i = 0041$_H$

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 (42_H)

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) = 0001_H
i = 0042_H

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**

H	0	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	ln										
8	BS		8	H	X	lng										
9	CAPS		9	I	Y	1/x										
A	2nd F		*	J	Z	↑										
B	DEL		+	K	=	EXP										
C	C-CE	▶	.	L	:	y^x										
D	⌟	◀	−	M	(	$\sqrt{\ }$										
E	↕	↑	.	N	)	x^2										
F	↑ Auto power OFF	↓	/	O		+/−										

Key reading (43_H)

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) = 0001_H
i = 0043_H
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

il = 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)

Keyed-in data \ Input	① [A]	② [CTRL] + [off]
1st byte	41_H	~ 0 H
2nd byte	00_H	$0F_H$

→ NEXT PAGE !

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

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

Data set to keyboard buffer (44_H)

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) $=0001_H$
i $=0044_H$
x =lead address of set data
a =number of set data bytes
return
nil

Key clear (45_H)

Clear the key scan buffer and keyboard buffer.

entry
(cx) $=0001_H$
i $=0045_H$
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 00_H code.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	CTRL hyp						CTRL 0	CTRL SPACE		RCL		SHIFT RCL		CTRL RCL		
1							CTRL 1	CTRL :	sin	STO	$\sin^{-1}$	SHIFT STO	CTRL sin	CTRL STO		
2				SHIFT INS		CTRL INS	CTRL 2	CTRL (	cos		$\cos^{-1}$	→xy	CTRL cos			
3			SHIFT ON BRK		CTRL ON/BRK	CTRL PF1	CTRL 3	CTRL)	tan		$\tan^{-1}$	n!	CTRL tan			
4	OFF		SHIFT OFF			CTRL PF2	CTRL 4		FSE	sin h	TAB	$\sin h^{-1}$	CTRL FSE			
5	BASIC		AER		CTRL BASIC	CTRL PF3	CTRL 5		→HEX	cos h	→DEC	$\cos h^{-1}$	CTRL HEX			
6	menu		CAL		CTRL menu	CTRL PF4	CTRL 6		→DEG	tan h	→D.MS	$\tan h^{-1}$	CTRL DEG			
7	カナ		SHIFT カナ		CTRL カナ	CTRL PF5	CTRL 7		ln		e^x		CTRL ln			
8			SHIFT BS		CTRL BS		CTRL 8		log		10^x		CTRL log			
9	CAPS		SHIFT CAPS		CTRL CAPS		CTRL 9		1/x		→rθ		CTRL 1/x			
A							CTRL *		↕		SHIFT ↕		CTRL ↕			
B			SHIFT DEL		CTRL DEL		CTRL +		EXP		π		CTRL EXP			
C			CA	SHIFT ▶	CTRL C.CE	CTRL ▶	CTRL .		y^x		$\sqrt[x]{y}$		CTRL y^x			
D			SFIFT ⌐	SHIFT ◀	CTRL ⌐	CTRL ◀	CTRL −		$\sqrt{}$		$\sqrt[3]{}$		CTRL $\sqrt{}$			
E	↕	DEG	SHIFT ◆	SHIFT ↑	CTRL ◆	CTRL ↑	CTRL .		x^2		%		CTRL x^2			
F	CTRL OFF			SHIFT ↓		CTRL ↓	CTRL /		+/−		SHIFT +/−		CTRL +/−			

⇌ **Auto power** OFF

Display of function key (46_H)

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) = 0001_H
i = 0046_H

return

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

Parameter for key input device driver

- repeat wait ($0BFCBA_H$, 1 byte)
  Specify the time from pressing of key until start of repeat. Unit: 16 msec.

- repeat pitch ($0BFCBB_H$, 1 byte)
  Interval between two keys in repeat state. Unit: 16 msec

- auto power off time ($0BFCBC_H$, 2 bytes, Low, hi)
  Time for auto power off. Unit: approx. 0.5 sec

- $\emptyset BFCBE_H$ [SOFTINT]
  bit 7: break
  bit 6: low battery

- $\emptyset BFCBF_H$
  bit 7: repeat on
  bit 4: click on

- ($BFCCO_H$, 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.

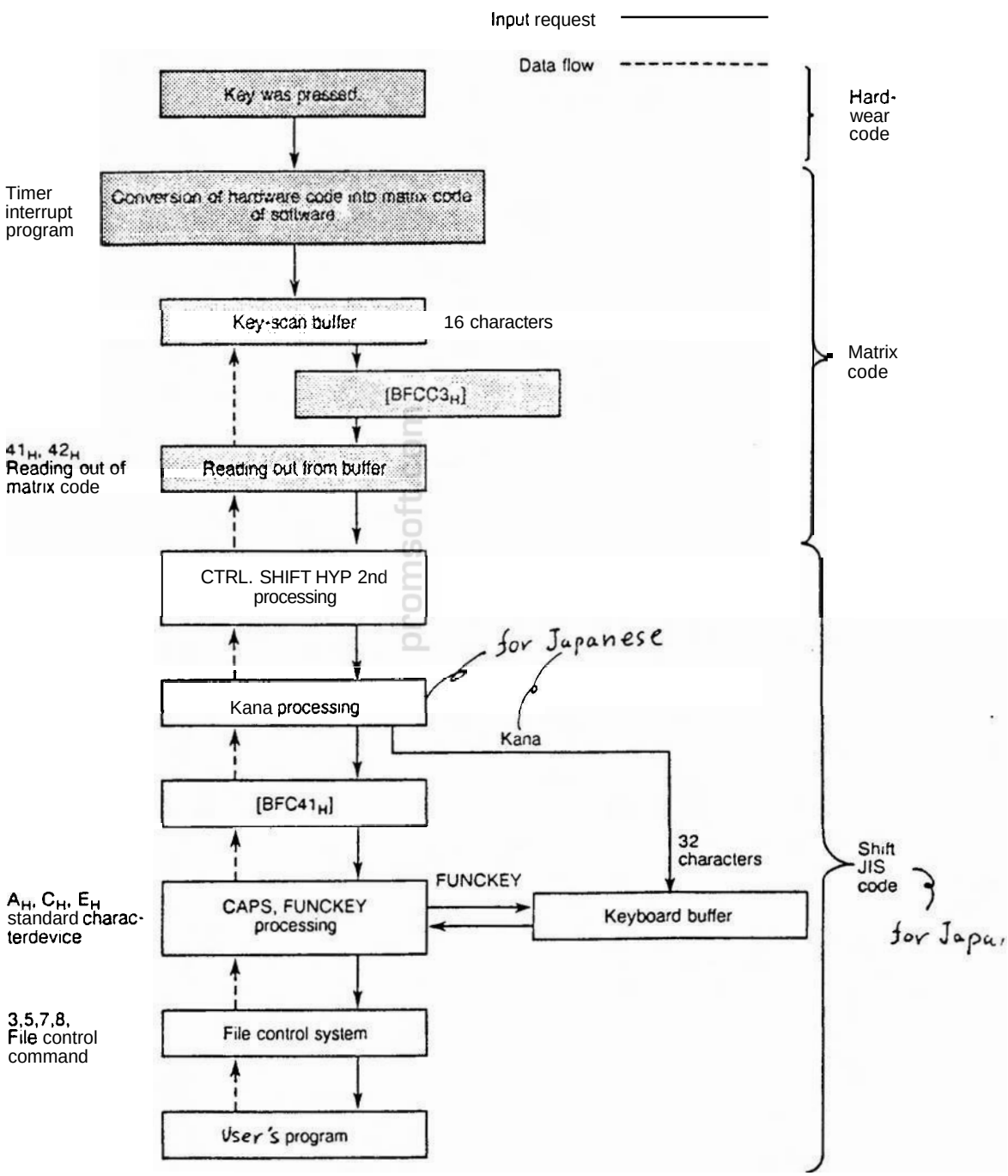
- ($BFCC3_H$, 3 bytes) Both are hook for key processing routine.
  ($BFC41_H$, 3 bytes)

- ($BFC45_H$, 1 byte)
  Slot number having function key data (FUNCKEY.).
  Ex: If it is 0, item in "S1:" is referred.

- ($BFC46_H$-$BFC51_H$, 9 bytes)
  File name having function key data.

- **Internal RAM FF_H**
  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 (25H). This is the same for verify (27H). 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 (40_H)

entry

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

1-byte direct output (41_H)

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

entry

(cx) = 0002_H
i = 41_H
a = output data

return

cy = 0: no error
cy = 1: error occured
a = 00_H : time out
FE_H : low battery
FF_H : break

1-byte direct input (42_H)

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

entry

(cx) = 0002_H
i = 42_H

return

cy = 0: no error
a = input data
cy = 1: error
a = 00_H: time out
FE_H: low battery
FF_H: break

Setting of hardware (43_H)

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

entry

(cx) = 0002_H
i = 43_H

return

nil

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

entry

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

return

nil

Reading of CS and CD ports ($4A_H$-$4B_H$)

entry

(cx) = 0002_H
i = $4A_H$: Specify CS port.
$4B_H$: Specify CD port.

return

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

Parameter Work

- SIO timer master $0BFD31_H$-$0BFD32_H$
  Time n on error timer ×0.5 (sec) However, $0FFFF_H$ is unlimited.
  default value = $0FFFF_H$ (unlimited)

- SIO baud rate $0BFD33_H$
  Specify baud rate, length, parity
  default value = $3C_H$ (00111100 B)

Bit 6, 5, 4 Baud rate		Bit 3, 2 Parity		Bit 1 Character length (Data)		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

$3C_H$: 1200 baud, Non-parity, 8-bit length, stop 1bit

Note

If the specified condition has been changed, '43_H' hardware set IOCS must be called.
Bit 7: 0

SIO Setup $0BFD34_H$
Specify shift in/out, X on/off. Specify transfer of transmission code at open/close.
Default value = 21_H
Bit 6 = 0: 1-byte data stored in 'SIO open send data' is not transmitted at open state.
1: Transferred 'SIO open send data' = $0BFD61_H$
Bit 4 = 0: 1-byte data stored in 'SIO close send data' is not transmitted at close state.
1: Transferred 'SIO close send data' = $0BFD62_H$
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 $0BFD35_H$
  Control of receive port
  Default value = 02_H
  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 $0BFD36_H$
  Control of receive port
  Default value = $0DF_H$
  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

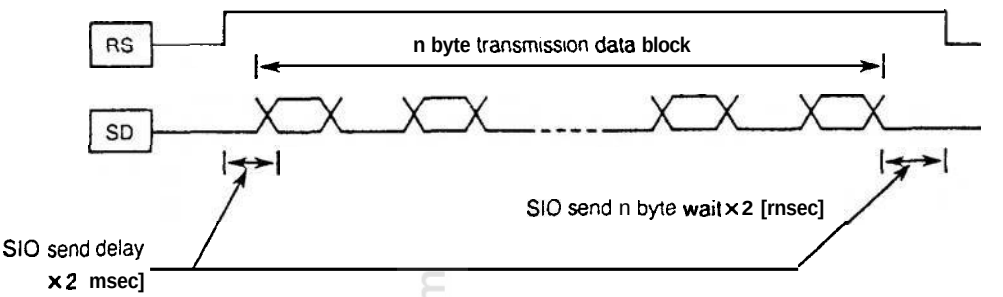
  Example
  $0DF_H$
  If RR-receiving buffer becomes full, the transmission side is stopped in RR Low.

- SIO send port condition $0BFD37_H$
  Control of SEND port
  Default value = 04_H
  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)
  ↳ $BFD31_H$ ~ 32H

- SIO send port control $0BFD38_H$
  Control of send port
  Default value = 50_H
  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 $0BFD39_H$
  [00-$0FF_H$] × 2 [msec] wait time is specified before or after transmission data block at transmission.
  Default value = 01_H [**2** msec]



- SIO crlf $0BFD3B_H$
  Specify the delimiter. External code is converted into internal delimiter ($0D_H + 0A_H$)
  Default value = 01_H
  Bit 1, 0: 00 = Not use
  01 = $0D_H$
  10 = $0A_H$
  11 = $0D_H + 0A_H$

Example
when specified the delimiter at $0D_H$ with 01, code is converted as follows.

		External	Internal E500
at sending	$0D_H + 0A_H \rightarrow 0D_H$	$0D_H$	$\leftarrow$ $0D_H + 0A_H$
at receiving	$0D_H \rightarrow 0D_H + 0A_H$	$0D_H$	$\rightarrow$ $0D_H + 0A_H$

- SIO eof code $0BFD3C_H$
  To specify the end code (external code is converted in internal end code ($1A_H$).)
  Default valuer $1A_H$
  Example
  When specified in end code 09_H, code is converted as follows.

		External	Internal E500
at sending	$1A_H \rightarrow 09_H$	09_H	$\leftarrow$ $1A_H$
at receiving	$09_H \rightarrow 1A_H$	09_H	$\rightarrow$ $1A_H$

- SIO open close wait $0BFD40_H$
  Wait n × 0.5 (msec) immediately after opening or immediately before closing.
  (for level converter CE-130T standby)
  Default value = 04_H (20 msec)

  Note
  Input of 00_H specifies 256 × 5 (msec)

- SIO open port control $0BFD41_H$
  Open of SIO port.
  Default value = 41_H
  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 $[00 \times 0FF_H] \times 2$ (msec) wait between send data 1 byte at sending.
  Default value = 00_H (non-wait)

- SIO open send data $0BFD61_H$
  Default value = 11_H
  When SIO setup bit-6 is 1, SIO open send data is transferred by 1 byte at open.

- SIO close send data $0BFD62_H$
  Default value = 13_H
  When SIO setup bit-4 is 1, SIO close send data is transferred by 1 byte at close.

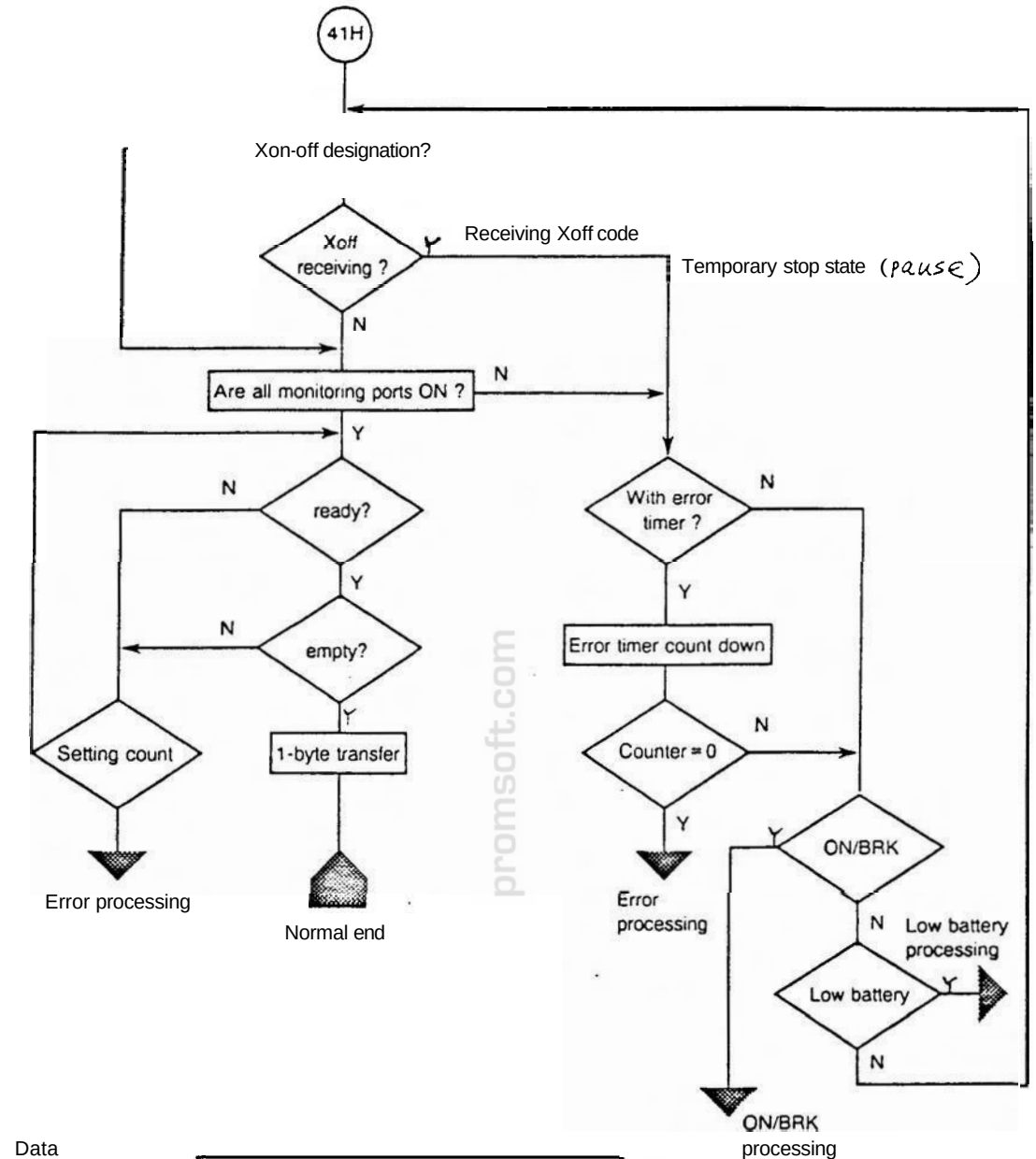
SIO hardware specifications

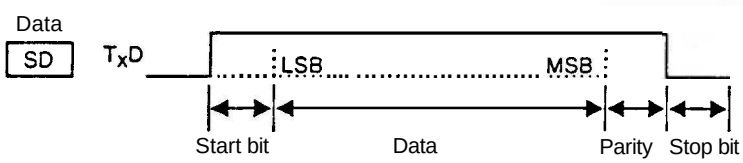
Baud rate:300, 600, 1200, 2400, 4800, 9600 (bps)
Data length: 7 or 8 bits
Stop bit :1 or 2 bits
Parity :even number, odd number, none
Duplex
Xon-off :possible to specify
Shift in/out: possible to specify

SIO software specifications

Transmission (1 byte)

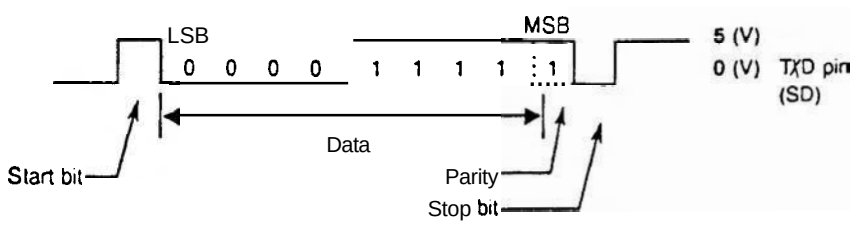
In case of Xon-off is specified, if Xoff code is being received, transmission side keeps waiting until X-code is received and released.
And, if the signal (port specified with SIO send port condition) to be monitored is not set ON (high level), it keeps waiting.
When above conditions are satisfied and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





Example

data=F0H , parity odd number, stop bit 1



X-off code	13_H	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	11_H	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-cut 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 (TXD)	Output	Data signal to be sent
3	Receive Data.	RD (RXD)	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 inpuffoutput 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 inpuffoutput 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 (40_H)

entry

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

Printing data output (41_H)

entry

(cx) = 0003_H
i = 41_H
a = output data

return

cy = 0: no error
= 1: error

Read of printing position (42_H)

entry

(cx) = 0003_H
i = 42_H

return

a = position of present printing head

Printer check (43_H)

entry

(cx) = 0003_H
i = 43_H

return

cy = 0: a = type of printer
i = number of max. printing digit
cy = 1: not connected
a = 54_H (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 20H - 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 = 44H: Output to tape
= 45H: 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 (40_H)

entry

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

return

nil

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

entry

(cx) = 0004_H
i = 41_H: Output to tape
42_H: Input from tape
43_H: 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 ($3F_H$)

All parameter are initialized.

entry

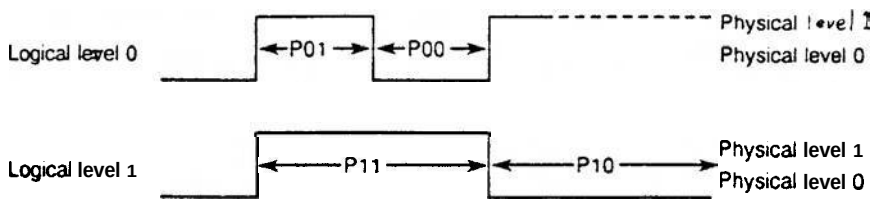
(cx) = 0004_H
i = $3F_H$

return

nil

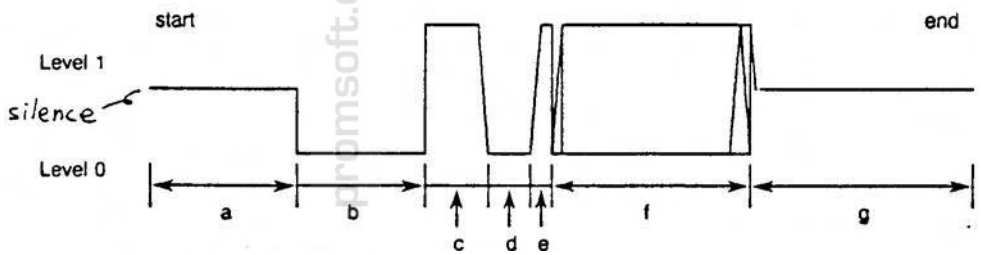
Parameter work

- Pulse length (baud rate can be changed.)
  ($BFD42_H$, 1 byte) length of logical level 0 and physical level 1 (P01)
  ($BFD43_H$, 1 byte) length of logical level 0 and physical level 0 (P00)
  ($BFD44_H$, 1 byte) length of logical level 1 and physical level 1 (P11)
  ($BFD45_H$, 1 byte) length of logical level 1 and physical level 0 (P10)
  ($BFD46_H$, 1 byte) threshold of logical level 0 and physical level 1



Following 0 and 1 should be logical level.

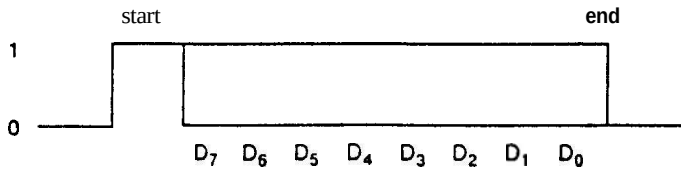
- Header block and data block



	a	b	c	d	e	f	g
Header block at write	[$BFD48_H$]	[$BFD4A_H$ 2 byte]	[$BFD4C_H$]	[$BFD4D_H$]	1	30 byte + α	[$BFD49_H$]
Header block at read	more than '0'	more than [BFD53H, 2byte]	[$BFD4C_H$]	[$BFD4D_H$]	1	30 byte + α	more than '0'
Data block at write	0	[$BFD4A_H$ 2 byte]	[$BFD50_H$]	[$BFD51_H$]	1	0 or more arbitrary byte + α	[$BFD52_H$]
Data block at read	more than '0'	more than [BFD53H, 2byte]	[$BFD50_H$]	[$BFD51_H$]	1	0 or more arbitrary byte + α	more than '0'

α : 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 30_H bytes with following composition.

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

The most top is the discriminator. 04_H 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 $1A_H$.

- Structure of CSAVE header block

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

(DI, Dh, De) = Data size

- Structure of CSAVEM header block

	+0		+2		+4		+6		+8		+A		+C		+E		
+00	01	File name									20	20	20	20	00	00	
+10	00	CD	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 ($3F_H$)

(file)

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 (40_H)

entry

(cx) = 0005_H
i = 40_H
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
  08_H-$0F_H$ Error

- Standard block device command
  10_H-$1F_H$ Error

- Special device command
  20_H-$2F_H$ Supported

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

Initialization of memory driver (40_H)

entry

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

return

nil

Format ($3F_H$)

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

entry

(cx) = 0006_H
i = $3F_H$
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 = $0A_H$: syntax error
$3C_H$: memory error

Search for physical address (41_H)

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

entry

(cl) = 06_H
(ch) =slot number (01112)
i = 41_H
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 = 01_H: no specified slot.
04_H: no block is found.

Condense (47_H)

Fill the space in **each** memory block.

entry

(cl) = 06_H
(ch) = slot number
i = 47_H

return

nil

Change of block size (42_H)

entry

(cl) = 06_H
(ch) = slot number
i = 42_H
a = free area pointer number (011)
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 = 00_H: slot (card) is protected.
01_H: no specified slot.
04_H: block is not found.
$0C_H$: insufficient memory
y = size possible to change
05_H: block is protected.

Rename of block (44_H)

Change the block name.

entry

(cl) = 06_H
(ch) = slot number of old block
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 = 00_H: slot (card) is protected.
01_H: no specified slot.
04_H: block is not found.
05_H: block is protected.
09_H: the same block name exists.

Transfer of block (43_H)

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

entry

(cx) = 0006_H
i = 43_H
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 (45_H)

Create new memory block.

entry

(cl) = 06_H
(ch) =slot number
i = 45_H
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 = 00_H: slot (card) is protected.
01_H: no specified slot.
09_H: the same block name exists.
$0C_H$: insufficient memory

Deletion of memory block (46_H)

Delete the memory block.

entry

(cl) = 06_H
(ch) = slot number
i = 46_H
x = address of block name

return

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

Creation of memory block 2 (48_H)

Create the memory block at the lead of the blocks

entry

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

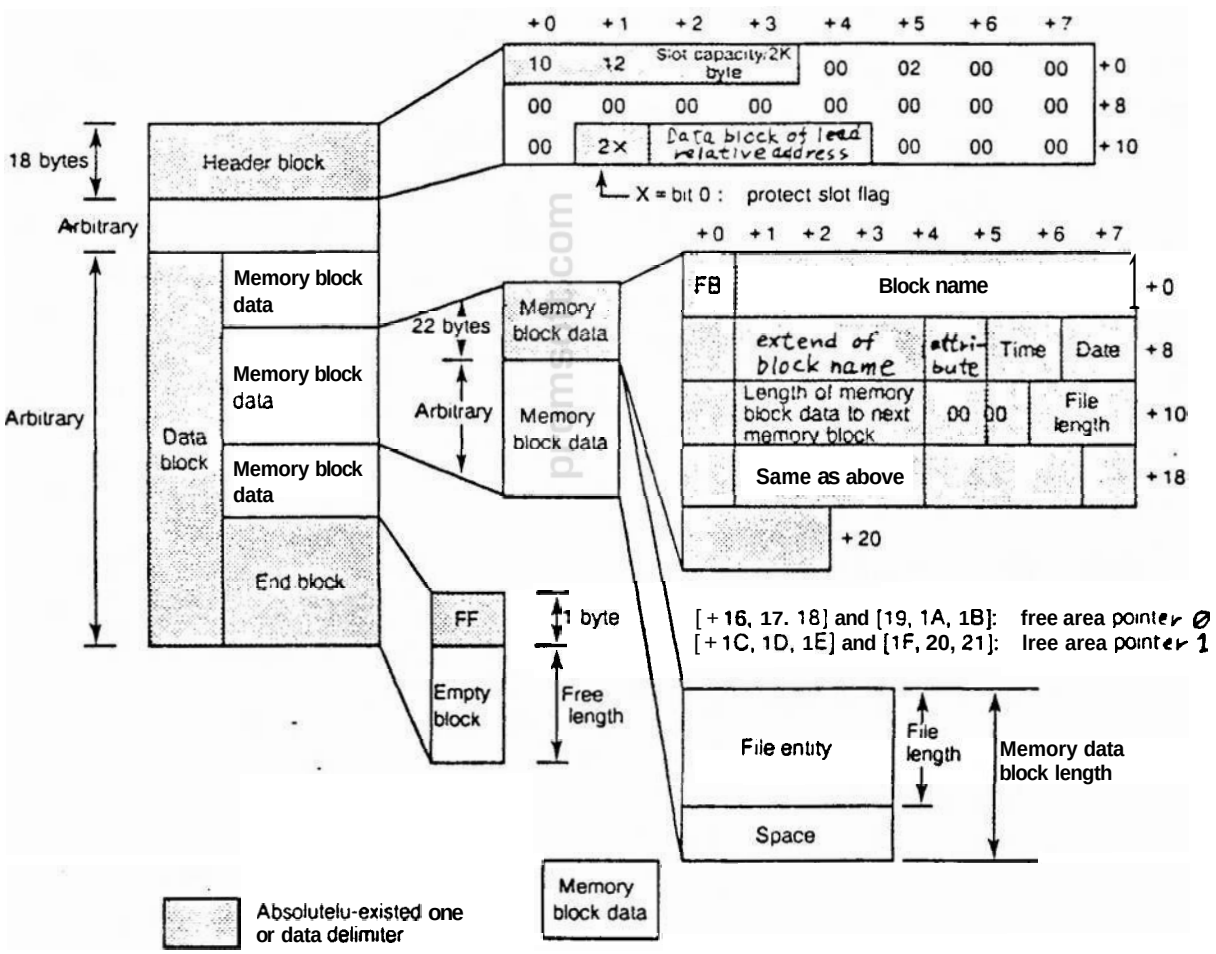
return

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

Parameter of memory block device

($BFC09_H$, 3 bytes)	lead address of slot 2 (S3:)
($BFC0C_H$, 2 bytes)	capacity of slot 2/2K bytes (S3:)
($BFC0F_H$, 3 bytes)	lead address of slot 1 (S2:)
($BFC12_H$, 2 bytes)	capacity of slot 1/2K bytes (S2:)
($BFC15_H$, 3 bytes)	lead address of slot 0 (S1:)
($BFC18_H$, 2 bytes)	capacity of slot 0/2K bytes (S1:)
($BFCDE_H$, 3 bytes)	last addres 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_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	no processing
41_H	power OFF
42_H	secure of work
43_H	execution of BASIC
44_H	(polynomial evaluation)
45_H, 46_H	Obtain processing address from the intermediate code.

Power off(41_H)

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

entry

(cx) = 0008_H
i = 41_H

return

nil

Secure the work (42_H)

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) = 0008_H
i = 42_H
x =work area pointer
$BFCDE_H$: for U stack
$BFCE1_H$: for S stack
$BFCE4_H$: reserve
$BFCE7_H$: reserve
$BFCEA_H$: reserve
$BFCED_H$: reserve
$BFCF0_H$: reserve
$BFCF3_H$: reserve
$BFCF6_H$: reserve
$BFCF9_H$: reserve
$BFCFC_H$: reserve
$BFCFF_H$: reserve
$BFD02_H$: reserve
$BFD05_H$: reserve
$BFD08_H$: reserve
$BFD0B_H$: reserve
$BFD0E_H$: BASIC work
$BFD11_H$: reserve
$BFD14_H$: reserve
$BFD17_H$: IOCS work
$BFD1A_H$: machine language area
y =size you want to secure

return

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

Execution of BASIC (43_H)

Execute the specified intermediate language character string.

entry

(cx) = 0008_H for BASIC
i = 43_H
x = intermediate language character string address
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 (45_H, 46_H)

entry

(cx) = 0008_H
i = 45_H: When command address is needed.
46_H: 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
  $1O_H$-$1F_H$ error

  Special device command
  20_H-28_H error

entry

(cy) = A
(ch) = B
i = C
when adding D and E,
[BFE03H] = D
[BFE02H] = 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 → X	9	0	47H		
		Subtraction Y-X → X			48H		
		Multiplication Y*X → X			49H		
		Division Y/X → X			4AH		
		Power Y^X → X			4BH		
	1-variable function	EXP e^X→X			4CH		
		SIN sinX→X			4DH		
		COS cosX→X			4EH		
		TAN tanX→X			4FH		
		SIN-1 $\sin^{-1}$X→X			50H		
		COS-1 $\cos^{-1}$X→X			51H		
		TAN-1 $\tan^{-1}$X→X			52H		
		DEG X→DEG→X			53H		
		DMS X→DMS→X			54H		
		ABS |X|→X			55H		
		INT intX→X			56H		
		SGN sgnX→X			57H		
		RND rndX→X			58H		
		SQR $\sqrt{X}$→X			59H		
		LOR logX→X			5AH		
		LN lnX→X			5BH		
Comparison	Numerical value	Y< >X			41H		
		Y<X			42H		
		Y>X			43H		
		Y=X			44H		
		Y≦X			45H		
		Y≧X			46H		

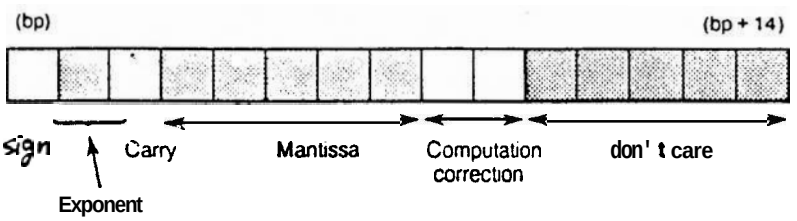
Function			A	B	C		
Comparison	Character	Y< >X	9	0	70H		
		Y<X			71H		
		Y>X			72H		
		Y=X			73H		
		Y≦X			74H		
		Y≧X			75H		
Conversion	Decimal → Binary conversion				7EH		
	Binary → Decimal conversion				7FH		
Character string operational function	ASC				76H		
	CJR $ CHR $				77H		
	STR $				78H		
	VAL				79H		
Matrix operation	Addition		9	1	41H		
	Subtraction				42H		
	Multiplication				43H		
	Division				44H		
	Inverse				45H		
	Addition to scholar				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		0-8 Y sequence
	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 $\pm 9.999999999 \times 10^{-99}$ and 0 with combination of exponent, mantissa sign and mantissa.



Zero is used when the mantissa is positive.
Eight is used when the mantissa is negative.

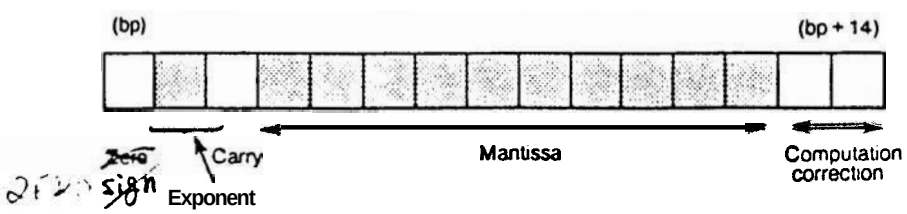
Exponent
The exponent is expressed using hexadecimal. The negative numbers are expressed using a complement.
9D (10^{-99})-63 (10^{-99})

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 $\pm 9.9999999999999999 \times 10^{99}$ and 0.



Mantissa sign

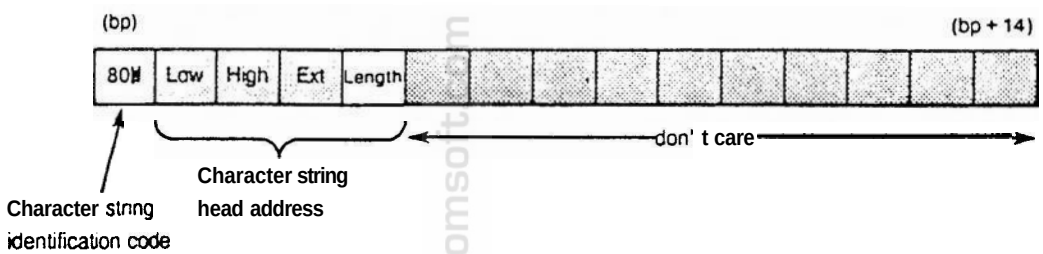
1 is used when the mantissa is positive.
9 is used when the mantissa is negative.

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.

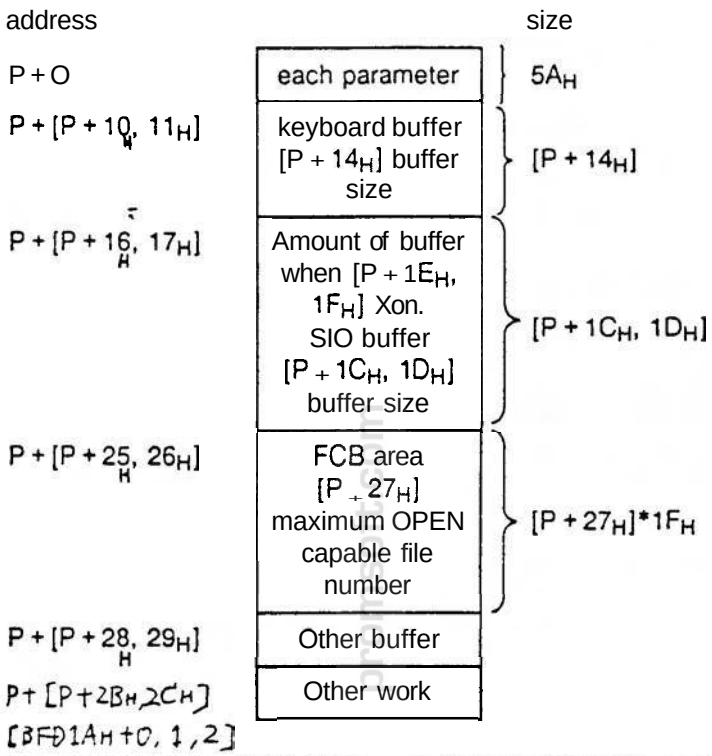
Pointer informations are shown in the figure.
Example: Refer the following program. Set the maximum opencapable file number at 10.
Execute as: (1) Input program
(2) RUN
(3) When the menu appears, select BASIC mode and input GOTO*.

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 &FFFD8
60 EKD
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 $E6_H$ + 0, 1, 2) or [$BFD17_H$ + 0, 1, 2]



address	Meaning
[P + 8_H]	LCD display start offset value Set in LCD in every approx. 5 min.
[P + 9_H]	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 + B_H, C_H]	Relative position of cursor screen buffer
[P + E_H, F_H]	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	
CD		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 provid.

Skip number discrimination code, real number discrimination code

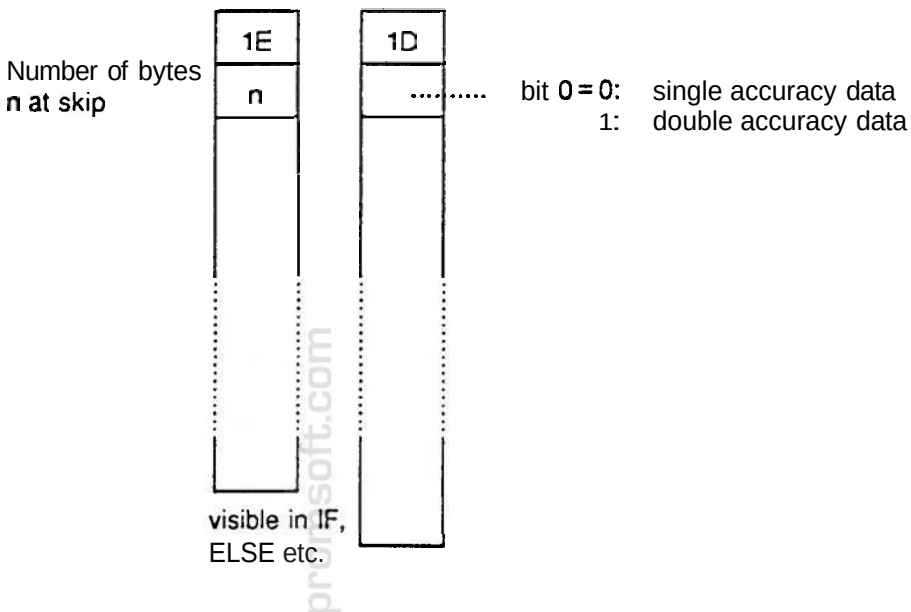


Table for BASIC intermediate code

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:DECI	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:AHS	9D:ASN
5E:DATA	6E:CROTATE	7E:	8E:AHC	9E:ACS
5F:PAUSE	6F:CIRCLE	7F:	8F:AHT	9F:ATN

A0:RND	B0:EOF	C0:ERN	D0:ASC	E0:	F0:CHR$
A1:AND	B1:DSKF	C1:ERL	D1:WAIT	E1:	F1:STR$
A2:OR	B2:LOF	C2:	D2:LEN	E2:	F2:HEX$
A3:NOT	B3:LOC	C3:	D3:	E3:	F3:
A4:PEEK	B4:	C4:	D4:	E4:	F4:
A5:XOR	B5:	C5:	D5:	E5:	F5:
A6:	B6:NCR	C6:	D6:	E6:	F6:
A7:EVAL	B7:NPR	C7:	D7:	E7:	F7:
A8:	B8:	C8:	D8:	E8:	F8
A9:	B9:	C9:	D9:	E9:INKEY$	F9:
AA:	BA:	CA:	DA:	EA:MID$	FA:
AB:	BB:	CB:	DB:	EB:LEFT$	FB:
AC:	BC:	CC:	DC:	EC:RIGHT$	FC:
AD:POINT	BD:	CD:	DD:	ED:	FD:
AE:PI	BE:AER	CE:	DE:	EE:	FE:
AF:FRE	BF:CUB	CF:	DF:	EF:	FF:

Intermediate code is expressed in 2 bytes for FE_H plus value in the above table.
For example, INPUT command is **stored** in **memory in the order of** FE_H, 81_H.

Location to store data

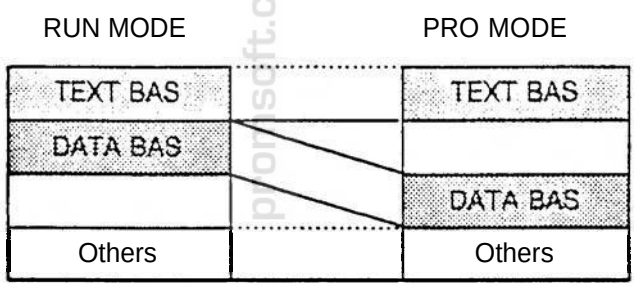
Data that may be necessary when operating BASIC are roughly classified as follows.

- 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
FUNCKN  .
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 out 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_H) 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_H) 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 ($0FC_H$)
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 ($0FB_H$)
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 ($0EB_H$)
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 $0FE_H$
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.
Wile 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.

$BFCC6_H$: fast timer interrupt
$BFCC9_H$: slow timer interrupt
$BFCCC_H$: key interrupt
$BFCCF_H$: ON key interrupt
$BFCD2_H$: SIO transmission interrupt
$BFCD5_H$: SIO reception interrupt
$BFCD8_H$: Outside interrupt
$BFCDB_H$: Software interrupt

Relation among isr, imr, and iisr operations