CHAPTER 11

Specifications of the PC-8401A Floppy Disk Driver

SECTION 1

OVERVIEW

DISK UNIT SUPPORTED BY FLOPPY DRIVER

The PC-8401 floppy disk driver supports the PC-8431A 3.5 inch floppy disk unit. Each of the two drives within provides for disk storage on an 80 track disk.

MEDIA FORMAT

The PC-8401 floppy disk driver uses the following physical disk format:

Item	*8431A*
Byte/Sector	256
Sector/Track	16
Track/Surface	80
Surface/Media	1

The CP/M format floppy disk is formatted in this way:

Block size	2048	bytes
Disk size	152	blocks
Directory entries	128	
System tracks	2	

The CP/M disk parameters are:

SPT	64
BLS	4
BLM	15
EXM	1
DSM	151
DRM	127
ALL0	0C0H
ALL1	000H
CKS	32
OFF	2

The PC-8401 CP/M does not use system tracks, even though two tracks are allocated as system tracks. This is to keep compatibility with the PC-8801A CP/M disk format (as used by the PC80312.DRV driver.) Also, future implementations of the 3.5 inch CP/M disks may require system tracks. Keeping the 3.5 inch disk logically compatible with current 5 inch disks simplifies media conversion.

DRIVE NUMBER

The PC-8401 floppy disk driver uses an absolute (fixed) drive numbering scheme. The first drive is always specified as 0, and the second drive as 1. This is not the same as the drive numbering convention used in conventional BDOS calls. The floppy drive numbering in the PC-8401A BDOS calls depends on its current CP/M mode. In the 32K mode, drive A: is always the built-in RAM disk, and the floppy disk drives, if any, are assigned the name B: (drive code 2) and C: (drive code 3). In the 64K mode, the floppy disk drives are named A: (drive code 1) and B: (drive code 2), when connected.

Application programs that use the PC-8401 floppy disk driver directly, rather than through BDOS calls, should note the differences in the drive numbering scheme.

In order to know which mode the application program is running, the program can look at the flag byte which is maintained by the PC-8401 bios and located at 0F7BBH (This address is named 'Cpmmode'). The Least Significant Bit of this flag will be 0 if the application program is running in the 32K mode, and 1 if running in the 64K mode.

Also note that floppy disk drives may not always be connected. To know how many drives are available, read the byte Flpnum at 0F7C1H. Flpnum=0 indicates that no floppy disk is available, and no floppy disk driver functions should be used.

SECTION 2

FUNCTIONS

READ SECTORS

Name Fread

Entry
C = 042H
A = Drive number (0 or 1)
B = Number of sectors to read (B <= 8)
D = Track number of 1st sector to read
E = Sector number of 1st sector to read
HL = Address where sector data are to be returned

Exit Cy set if I/O error is detected, or specified drive does not exist.

The Fread function reads one or more sectors into the user buffer. Up to 8 sectors can be read at a time. Only one track may be read (ie., track spanning on read is not supported by this function call.) Track numbers should range from 0 to 79, and sector numbers range from 1 to 16.

The Fread function performs a Block Read to improve disk I/O throughput. When Fread is requested to read just one sector, this function actually reads 8 sectors. If there are less than 8 sectors remaining on the track, this function will read from the specified sector to the last sector on the track (whichever is smaller.) The data is read into the internal buffer in the disk unit. The contents of the sector specified by DE is then passed to the user buffer. If the specified sector has already been read into the block, Fread does not perform a physical read, but instead gets the sector data from the internal buffer of the disk unit.

The data in the internal buffer is discarded in the following cases:

1. A Warm Boot occurs.
2. The Bios HOME call is made (Restore to track 0.)
3. A write operation to the floppy disk is performed.
4. A disk format operation is performed.
5. A Disk I/O error occurs.
6. The next sector read request didn't come in 10 seconds. (This is just to stop the drive motor because it keeps rotating while the sector data exists in the internal buffer.)

WRITE SECTORS

Name Fwrite

Entry
C = 043H
A = Drive number (0 or 1)
B = Number of sectors to write (B <= 8)
D = Track number of 1st sector to write
E = Sector number of 1st sector to write
HL = Address where sector data to be written is stored

Exit Cy set if I/O error is detected, or specified drive does not exist.

The Fwrite function writes up to 8 sectors at a time. Only one track may be written (ie. track spanning on write is not supported by this function call.) Track numbers should range from 0 to 79, and sector numbers will range from 1 to 16.

FORMAT MEDIA

Name Ffrmt

Entry
C = 044H
E = Drive number (0 or 1).

Exit Cy is set if an I/O error occurs, or the specified drive does not exist.

The Format function performs physical formatting of the floppy disk. Note that all sectors are filled with 0FFH, not 0E5H.

CHAPTER 12

Technical Documentation for the PC-8431A

NEC Corporation

SECTION 1

GENERAL DESCRIPTION

The PC-8431A is a micro floppy disk system which has its own CPU and memory buffer. It communicates with the PC-8401A host computer through a "3-wire handshake protocol". Since the PC-8431A is a stand-alone disk system, the host computer is free from the cumbersome control of a resident Floppy Disk Controller. The PC-8431A performs all the physical disk I/O operations upon receipt of command instructions and optional arguments, from the host, which select the operations to be performed by the PC-8431A.

The interchange protocol between the host and PC-8431A involves the specification of physical sectors on the disk. Any other sophisticated file handling is done by the host computer under its program control.

SECTION 2

OPERATION MODE

The operation mode can be either Single Track or Double Track. In the Single Track mode, only reads can be performed (the write operation is not supported.) Capacity for a Single Track mode disk is 150K bytes per drive.

Data can be read from or written to a Double Track mode disk. Capacity for a Double Track mode disk is 300K bytes per drive.

SECTION 3

COMMAND SUMMARY

INITIALIZE COMMAND

The Initialize command resets the uPD-765 Floppy Disk Controller, and recalibrates the heads in the PC-8431A. No parameter is required for this command.

COMMAND FORMAT
<INITIALIZE>
00H

WRITE DATA COMMAND

The Write Data command writes blocks of data which are sent from the host computer following the command. Each block has 256 bytes of data.

COMMAND FORMAT					
<WRITE>	<N>	<DD>	<TT>	<SS>	<DATA>
01H	1-8	0-1	0-79	1-16	xxxx

Where <N> is the number of blocks (1 to 8) to write that will follow this command as <DATA>, <DD> is the drive number (0 to 1) of the disk to write upon, <TT> is the track number (0 to 79) of the disk, and <SS> is the sector number (1 to 16) that the write begins upon. The maximum number of blocks that can be written by this command is 8. Data CANNOT be written across tracks, that is, <N> and <SS> must satisfy the following restriction:

<N> + <SS> must be less than or equal to 17

EXAMPLE: To write 512 bytes (2 blocks) of data on the 5th and 6th sectors of track 10 on drive 0, issue the following command and parameters:

01H	02H	00H	0AH	05H	<512 bytes of data>
<WRITE>	<N>	<DD>	<TT>	<SS>	<DATA>

Note that only Double Track mode disks may be written.

If an error is detected during the execution of a command which invokes disk I/O operation, the code in the PC-8431A drive attempts to recover from the error by the repeating the I/O operation several times.

READ DATA COMMAND

The Read Data command is used to read data from the specified disk's sectors into the buffer in PC-8431A. Note that the read data command does not send the data to the host. Instead, the data is stored in the buffer of the PC-8431A. The host must then issue the Send Data command after the completion of the Read Data command to get the data out of PC-8431A. By separating the read operation from the send operation, the host system can utilize a read-ahead buffering technique to improve disk I/O throughput.

COMMAND FORMAT

	<READ>	<N>	<DD>	<TT>	<SS>
Double Track	02H	1-8	0-1	0-79	1-16
Single Track	02H	1-8	0-1	0-39	1-16

Where <N>, <DD>, <TT>, and <SS> are identical to the Write Data command parameters and follow the same <N> + <SS> restriction. Note that if a Single Track mode disk is being read, the maximum value for <TT> is 39.

SEND DATA COMMAND

This command requests the PC-8431A to transmit the contents of the read buffer containing the data from the most recently executed Read Data command.

COMMAND FORMAT
<SEND>
03H

The length of data to be transmitted is <N>*256, where <N> is what was specified by the most recently executed Read Data command.

The Send Data command should only be issued after the prior Read Data command was successfully completed, otherwise the PC-8431A will hang up. The Send Result Status command should be invoked to inform the host computer whether a command was successfully completed or not.

COPY COMMAND

The copy command copies data from specified source sectors to specified destination sectors.

	COMMAND FORMAT							
				SOURCE		*DESTINATION*		
	<COPY>	<N>	<DD>	<TT>	<SS>	<DD>	<TT>	<SS>
Double Track	04H	1-8	0-1	0-79	1-16	0-1	0-79	1-16
Single Track	04H	1-8	0-1	0-39	1-16	0-1	0-39	1-16

Where <N>, <DD>, <TT>, and <SS> are identical to the Write Data and Read Data command parameters and follow the same <N> + <SS> restriction. Note that if a Single Track mode disk is being read, the maximum value for <TT> is 39.

FORMAT COMMAND

A new (factory fresh) diskette should be initialized using this command prior to its use as a data diskette. The Format command writes sector IDs and gap lengths on the diskette, and fills all the sectors with 0FFH.

COMMAND FORMAT	
<FORMAT>	<DD>
05H	0-1

Where <DD> specifies the drive number containing the diskette to be formatted.

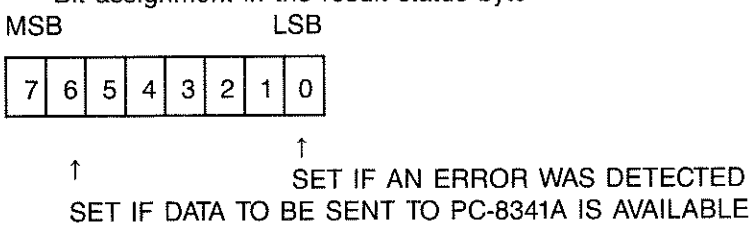
SEND RESULT STATUS COMMAND

After every command which involves a read/write operation, the Send Result Status command should be issued to find out whether the read/write command was successful. This command returns a one byte result code which shows whether data is available or not and if an error occurred or not.

The current version of the disk I/O code in the PC-8431A can accept a command only after the preceding command has been completed.

COMMAND FORMAT
<SEND RESULT STATUS>
06H

Bit assignment in the result status byte



There are several cases when an error is reported, and the Send Result Status command does not give enough information for determination of the cause of the reported error. The tool provided for such analysis, the Send FDC Result command, gives the host computer more detailed information about the errors.

SEND DRIVE STATUS COMMAND

This command gives drive status information. The host computer can know, through use of this command, whether a drive is ready (on-line) or not.

COMMAND FORMAT
<SEND DRIVE STATUS>
07H

Bit assignment in the result status byte

MSB LSB

7	6	5	4	3	2	1	0

↑ ↑

SET IF DRIVE 0 IS READY (ON-LINE)

SET IF DRIVE 1 IS READY (ON-LINE)

The hardware of the PC-8431A has no check mechanism for sensing whether a drive is on-line at any given time during it's operation. The status is only correctly returned after the disk drive head is recalibrated upon receipt of the Initialize command.

SEND FDC RESULT COMMAND

The Send FDC Result command is used by the host computer to obtain more detailed information about any error which was reported in response to the Read Result Status command. The Send FDC Result command returns the last FDC command's result status bytes.

Although the send FDC result command is designed mainly to analyze error conditions, it may be issued even after a floppy disk controller has been successfully completed.

COMMAND FORMAT
<SEND FDC RESULT>
09H

The Send FDC Result command sends 7 result status bytes:

1st	2nd	3rd	4th	5th	6th	7th
ST0	ST1	ST2	C	H	R	N

Where ST0, ST1 and ST2 are the three registers which store the status information after an FDC command has been executed, C is the cylinder (track) number, H is the head number as specified on the disk's ID field, R is the record (sector) number that was read or written, N is the number of data bytes written in the sector. These bytes are explained in more detail in the "uPD-765 SINGLE/DOUBLE DENSITY FLOPPY DISK CONTROLLER" specification sheet, available from NEC Microcomputers, Inc.

MARGIN PARAMETER SET COMMAND

The Margin Parameter Set command sets the data signal's window timing for the uPD765A Floppy Disk Drive controller.

COMMAND FORMAT	
<MARGIN PARAMETER SET>	<DATA>
0AH	0EH

This parameter is set by the ROM code of the PC-8431A unit. It must not be altered by an application program, as it's value is already preset for proper read timing by the firmware. Its setting is dependent upon the floppy disk drive specifications. If an application program alters the value, the FDD will misread the floppy disk data. (The Read Data Window timing is normally attached to a Phase Locked Loop at pin 22 of the uPD765 FDD.)

TRANSMIT ID DATA

The Transmit ID Data command was designed into the PC-8431A interchange protocol for future implementation. Its purpose will be to provide data start addresses and volume numbers to the host computer. Currently, the PC-8431A ignores this command and simply returns the byte 0EFH to the host.

COMMAND FORMAT
<TRANSMIT ID DATA>
0BH

The current implementation sends 0EFH to the host computer. Future implementations will return four bytes:

<HA> <LA> <HV> <LV>

where <HA> and <LA> are the high and low addresses, and <HV> and <LV> are the high and low bytes of the data volume. This command will be issued when the host computer requires a signature byte.

DIRECT SEEK COMMAND

The Direct Seek command moves the head to (seeks) the specified track number on the disk.

COMMAND FORMAT			
	<SEEK>	<DD>	<TT>
Double Track	0CH	0-1	0-79
Single Track	0CH	0-1	0-39

DIRECT RECALIBRATE COMMAND

The Direct Recalibrate command recalibrates the head (restores the head to track 0.)

COMMAND FORMAT	
<RECALIBRATE>	<DD>
0DH	0-1

TEST MODE ON COMMAND

The Test Mode On command is used to set the test mode on. This is provided for maintenance purposes and should not be requested by application programs. Standard application programs require the automatic retry that is built into the interface protocol code. When the Test Mode is on, retries do not occur. In the Test Mode, during a read or write operation, the interface protocol simply terminates the operation when an error is detected.

COMMAND FORMAT
<TEST MODE ON>
0EH

TEST MODE OFF COMMAND

The Test Mode Off command is used to set the test mode off. The interface protocol will attempt automatic retry and recovery of errors.

COMMAND FORMAT
<TEST MODE OFF>
0FH

FAST WRITE COMMAND

The Fast Write command is identical to the Write Data command described previously, except that the Fast Write command uses the fast handshake protocol (two bytes per handshake cycle) to pass the data bytes. The command byte and arguments are to be sent in the normal fashion. The same restrictions as in the Write Data command apply here.

COMMAND FORMAT					
<FAST WRITE>	<N>	<DD>	<TT>	<SS>	<DATA>
011H	1-8	0-1	0-79	1-16	xxxx

FAST SEND COMMAND

The Fast Send command is identical to the Send command except that the Fast send command uses the fast handshake protocol to send the data bytes to the host computer. The command byte and arguments are to be sent in the normal fashion.

COMMAND FORMAT
<FAST SEND>
012H

SET OPERATION MODE COMMAND

The set operation mode command sets the operation mode, previously described.

In the single track mode, the read operation is supported, but write operations are not available. The Write Data command, the Format command, the Copy command and the Fast Write command cannot be used.

In the double track mode, all operations are supported.

COMMAND FORMAT	
<SET OPERATION MODE>	<SET BYTE>
017H	0-1

Bit assignment in the result status byte:

MSB LSB

7	6	5	4	3	2	1	0

↑
RESET FOR SINGLE TRACK MODE
↑
SET FOR DOUBLE TRACK MODE

SECTION 4

COMMAND CODE ASSIGNMENTS SUMMARY

INITIALIZE COMMAND	00H
WRITE DATA COMMAND	01H
READ DATA COMMAND	02H
SEND DATA COMMAND	03H
COPY COMMAND	04H
FORMAT COMMAND	05H
SEND RESULT STATUS COMMAND	06H
SEND DRIVE STATUS COMMAND	07H
SEND FDC COMMAND	09H
MARGIN PARAMETER SET COMMAND	0AH
TRANSMIT ID DATA	0BH
DIRECT SEEK COMMAND	0CH
DIRECT RECALIBRATE COMMAND	0DH
TEST MODE ON COMMAND	0EH
TEST MODE OFF COMMAND	OFH
FAST WRITE COMMAND	011H
FAST SEND COMMAND	012H
SET OPERATION MODE COMMAND	017H

All values are in hexadecimal. If invalid command is given, the PC-8431A simply ignores it.

SECTION 5

THREE WIRE HANDSHAKE PROTOCOL

Data transfer between the host computer and the PC-8431A is performed by means of a three wire handshake hardware protocol. This protocol is similar to the IEEE-488 handshake hardware protocol. In the handshaking, three control lines are used; RFD (ready for data), DAV (data valid), and DAC (data accepted).

The RFD is set true by the receiver to tell the sender that it is ready to receive a new byte. The DAV is set true by the sender when a new byte is put on the bus to inform the receiver that the data on the bus is valid and the receiver may get the byte. The DAC is set true by the receiver when the data is received from the data bus.

The DAV line should be held true until the receiver sets the DAC true. The receiver resets the DAC line false as soon as the receiver discovers that the DAV line has gone false. From the receiver's point of view, one handshake cycle is complete when the DAV line went to false and the receiver reset the DAC line in response. From the viewpoint of the sender, the handshake cycle ends when the sender discovered that the DAC line was reset in response to the sender reseting the DAV line.

Another control line, ATN (attention), should be set by the host computer only when the command byte is sent. It is reset when arguments or data bytes are sent. The ATN line is used to confirm the synchronization between the host computer and the PC-8431A. Through use of this line, sychronization is kept in the case where the host computer aborts a command during the transmission of the command/argument/data string to the PC-8431A, or during the reception of the data/status bytes from the PC-8431A. If such an abort should occur, by setting the ATN line true, the current process within the PC-8431A is aborted, and the PC-8431A returns to the initial state of reception and interpretation of the new command byte.

Some example handshake routines are outlined below. An 8255 chip is used in the host computer to physically connect it to the PC-8431A. The following routines assume that an 8255 is used.

```
;
;   SAMPLE PROGRAM FOR HANDSHAKE
;
```

```
;Bit assignment
MY__RFD     EQU 5               ;PC-8401A's RFD line is 5th bit
                                ; in port C counting from 0
MY__DAV     EQU 4
MY__DAC     EQU 6
MY__ATN     EQU 7

ITS__RFD    EQU 00000010B       ;PC-8431A RFD line.
                                ;This is the bit mask to pick
                                ;up the bit out of port C.
ITS__DAV    EQU 00000001B
ITS__DAC    EQU 00000100B

PORTA       EQU 0FCH            ;Where we receive data byte
PORTB       EQU 0FDH            ;Where we send data byte
PORTC       EQU 0FEH            ;Where we read or write control signals

CW          EQU 0FFH            ;Command port of 8255
```

```
;
;Sender handshake routine for the host computer to send a command
;byte to the PC-8431A. The ATN line is set by this routine.

SHCMD:
        PUSH AF         ;Save the command byte
        LD A,MY_ATN*2+1
        OUT (CW),A      ;Set ATN line using the bit
                        ;set/reset instruction of the 8255
        POP AF          ;Recall the command byte
                        ;Then fall into the "SH" routine

;Sender handshake routine for host computer
;
;This routine should be used to send an argument or data byte.
;The command byte should be sent by the SHCMD routine above.
;The data byte in [A] is sent to the PC-8431A.

SH:
        PUSH AF         ;Save data byte to be sent
NOTRDY:
        IN A,(PORTC)    ;All control lines are connected
                        ;in port C of the 8255
        AND ITS_RFD     ;See if the PC-8431A is ready
        JRZ NOTRDY      ;Wait for RFD to go high

        LD A,MY_ATN*2   ;It is now ready
                        ;Reset ATN line
        OUT (CW),A
        POP AF          ;Recall data byte we will send
        OUT (PORTB),A   ;Output data bus is on port B

        LD A,MY_DAV*2+1 ;Set data valid true
        OUT (CW),A      ;Use bit set/reset instruction of 8255
NOTACP:
        IN A,(PORTC)    ;See if it has received data
        AND ITS_DAC
        JRZ NOTACP      ;Wait if not

        LD A,MY_DAV*2   ;It has received
                        ;Reset MY_DAV line
        OUT (CW),A
STLDAC:
        IN A,(PORTC)    ;Then see if it has reset the DAC line
        AND ITS_DAC
        JRNZ STLDAC     ;Still true, wait for the DAC to go false
        RET             ;All done
```

```
;Receiver handshake routine
;
;Data byte sent from PC-8431A is received in [A]
;
AH:
            LD A,MY__RFD*2+1    ;Tell PC-8431A that I'm ready
            OUT (CW),A
NOTDAV:
            IN A,(PORTC)        ;Wait for DAV line to go true
            AND ITS__DAV
            JRZ NOTDAV

            LD A,MY__RFD*2      ;Now data on the bus is valid
                                ;Reset my RFD soon
            OUT (CW),A

            IN A, (PORTA)       ;Receive data byte through port A
            PUSH AF             ;Save it
            LD A,MY__DAC*2+1    ;Tell it that I have received data
            OUT (CW),A
STLDAV:
            IN A,(PORTC)        ;Wait for DAV line to go false
            AND ITS__DAV
            JRNZ STLDAV         ;DAV is still true

            LD A,MY__DAC*2      ;DAV is gone false, reset MY__DAC
            OUT (CW),A

            POP AF              ;Handshake has completed
            RET
```

The operating mode of the 8255 should be fixed prior to the first handshake by sending the command instruction 91H to the command port.

The fast handshake protocol is very similar to the normal handshake protocol described above. The difference is that in the fast handshake protocol, two bytes are sent or received in one handshake cycle. One byte is sent or received at the leading edge of the DAV line, while another byte is sent or received at the trailing edge.

The two routines listed below, "FASTCH" and "FASTAH" are sample fast handshake routines. Note that there is no fast handshake routine to send the command byte. As described in the previous sections, the command byte and arguments are always sent by using the normal handshake protocol. Only data bytes are sent or received through the fast handshake routines.

```
;Fast sender handshake routine
;
;Data bytes to be sent are expected to be stored in a buffer,
;with [HL] pointing to the 1st byte to be sent.
;
;On exit, [HL] is incremented by 2

FASTSH:
            IN A,(PORTC)        ;Is PC-8431A is ready for new handshake?
            AND ITS_RFD
            JRZ FASTSH          ;No, wait

            LD A,(HL)           ;Its ready. Pick up a data byte
                                ;to be sent
            INC HL              ;Bump data pointer
            OUT (PORTB),A       ;Output the byte on bus

            LD A,MY_DAV*2+1     ;Tell it that I put a data
                                ;byte on bus
            OUT (CW),A
NOTDAC:
            IN A,(PORTC)        ;Wait for it to say it's received
                                ;the first byte
            AND ITS_DAC
            JRZ NOTDAC

            LD A,(HL)           ;It's received the first byte
                                ;Let's send the second byte
            INC HL
            OUT (PORTB),A

            LD A,MY_DAV*2       ;Reset DAV to say that the
                                ;second byte is now on the bus
            OUT (CW),A
DACTRUE:
            IN A, (PORTC)       ;Then wait for DAC to go false which
                                ;indicates that the PC-8431A has
                                ;received the second byte
            AND ITS_DAC
            JRNZ DACTRUE
            RET                 ;All done
```

```
;Fast receiver handshake routine
;
;Received two bytes are stored where [HL] points
;to on entry

FASTAH:
        LD A,MY__RFD*2+1    ;Tell PC-8431A I'm ready
        OUT (CW),A
DAVLOW:
        IN A,(PORTC)        ;Wait for DAV to go true
        AND ITS__DAV
        JRZ DAVLOW

        LD A,MY__RFD*2      ;DAV is true. Tell it that I'm busy
        OUT (CW),A

        IN A,(PORTA)        ;Receive the first byte
        LD (HL),A           ;Save it
        INC HL

        LD A,MY__DAC*2+1    ;Set DAC true to tell it
        OUT (CW),A          ;I have received first byte
DAVHIGH:
        IN A,(PORTC)        ;Then wait for the send byte
                            ;to be available
        AND ITS__DAV
        JRNZ DAVHIGH

        IN A,(PORTA)        ;DAV flipped again
                            ;Now 2nd byte is available
        LD (HL),A
        INC HL

        LD A,MY__DAC*2      ;Reset DAC to say that
                            ;I have received both bytes
        OUT (CW),A

        RET                 ;All done
```

CHAPTER 13

Specifications of the MODEM7 Protocol in TELCOM

NEC Corporation

The following expresses the control flow for a file transfer using the MODEM7 protocol. MODEM7 was originally conceived and written by Ward Christensen. It was written as a tool for the transfer of CP/M programs and data, from computer to computer through a "Null-Modem" cable or, by using a modem, over telephone lines. The protocol provides reliable transfer, with error checking of each 128 byte data block sent, by utilizing a checksum.

The protocol imposes no restrictions on the contents of the data being transmitted. No control characters are looked for in the 128 byte data messages. Absolutely any kind of data may be sent: binary, ASCII, etc. The parameters used are: Word length - 8 bits, Parity - none, Stop bits - 1. Any Baud rate may be used with this protocol. The most common rates are 300, 1200, 2400 (and 9600 in a direct connection configuration).

Those wishing to maintain compatibility of the CP/M file structure, i.e. to allow the transfer of ASCII files to or from CP/M systems, should follow this data format:

1. ASCII tabs used (09H); tabs set at 8 column intervals.
2. Lines terminated by CR/LF (0DH, 0AH)
3. End of file indicated by ^Z, 1AH. (one or more)
4. Data should be considered as a continuous stream of data bytes, broken into 128-byte chunks, purely for the purpose of transmission.
5. If the data ends exactly on a 128-byte boundary, i.e. CR on byte # 127, and LF on byte # 128, a subsequent sector containing the ^Z EOF character(s) is preferred. Some utilities or user programs cannot handle EOF without ^Zs.
6. The last block sent is no different from others., i.e. there is no "short block".

The following ASCII codes are used to structure the protocol:

Start of Header	<SOH>	01H
End of Transmission	<EOT>	04H
Acknowledgement	<ACK>	06H
Negative acknowledgement	<NAK>	15H
Cancel	<CAN>	18H

Transmitting

Each block of the transfer takes this format:

<SOH> <BLK#> <255-BLK#> <--128 DATA BYTES--> <CKSUM>

in which:

<SOH>	Start of Header character.
<BLK#>	Binary number, starts at 001 increments by 001, and wraps at 0FFH back to 000H.
<255-BLK#>	The complement of BLK#, ie. each bit in the 8-bit block number is complemented.
<CKSUM>	The sum of the data bytes only. Any carries are ignored.
<CAN>	is used for canceling the transmission.

All errors are retried 10 times. A message is typed after 10 retries asking the operator whether the program should retry or quit.

The receiver has a 10-second timeout. It sends a <NAK> after each timeout. The receiver's first <NAK> signals the transmitter to start. The receiver must continue to timeout every 10 seconds in case the sender isn't ready.

Once into receiving a block, the receiver goes into a one second timeout for each character and the block-ending checksum. If the receiver wishes to <NAK> a block for any reason (invalid header, timeout receiving data), it must wait for the line to clear.

Synchronizing

If a block number is received, it will be either:

1) The expected one, in which case everything is fine . . . or
2) A repeat of the previously received block. This should be considered as being okay. It only indicates that the receiver's <ACK> became glitched, and the sender retransmitted . . . or
3) Any other block number, which indicates a fatal loss of synchronization, such as the rare case of the sender getting a line glitch that looked like an <ACK>. Abort the transmission, sending a <CAN>.

Sending

While waiting for the receiver to send the initial <NAK> indicating transmission is to begin, the sender has one very long timeout of about 45 seconds. Once in the sending protocol, the sender has a 10 second timeout before it retrys.

When the sender has no more data, it sends an <EOT>, and awaits an <ACK>, resending the <EOT> if it doesn't get one.

Example

Here is a sample of the data flow, sending a 3-block message. It includes the two most common transmission interruptions - a garbaged block, and an <ACK> reply getting garbaged. <XX> represents the checksum byte.

SENDER		RECEIVER
		Timeout after 10 seconds,
	←	<NAK>
<SOH> 01 FE -DATA- <XX>	→	
	←	<ACK>
<SOH> 02 FD -DATA- <XX>	→	(DATA gets a line glitch)
	←	<NAK>
<SOH> 02 FD -DATA- <XX>	→	
	←	<ACK>
<SOH> 03 FC -DATA <XX>	→	
(<ACK> gets garbaged)	←	<ACK>
<SOH> 03 FC -DATA- <XX>	→	
	←	<ACK>
<EOT>	→	
	←	<ACK>

CHAPTER 14

Specifications of the ROM in the PC-8401A

NEC Corporation

SECTION 1

OUTLINE

In the PC-8401, two different types of ROM are supported. The first is called Mapped ROM and the second is called I/O ROM. The Mapped ROM, addressed from 00000H to 07FFFH, can be accessed directly by the CPU when selected. (Memory management is discussed elsewhere in this manual.) On the other hand, the I/O ROM cannot be accessed directly by the CPU, but only through I/O (Input and Output) instructions.

Multiple CP/M programs can be stored both in the Mapped ROM and in the I/O ROM. The names of the programs in the ROMs always appear at the top of the Menu Screen, followed by the user application files and data files in the selected file storage device. When Cold or Warm Boot occurs and the Menu screen is displayed, the directory area in ROM is searched as part of the start-up routine. The file names in ROM are obtained much like those in the file storage area in the RAM file and on the Floppy Disk.

SECTION 2

NUMBER OF FILES IN ROM

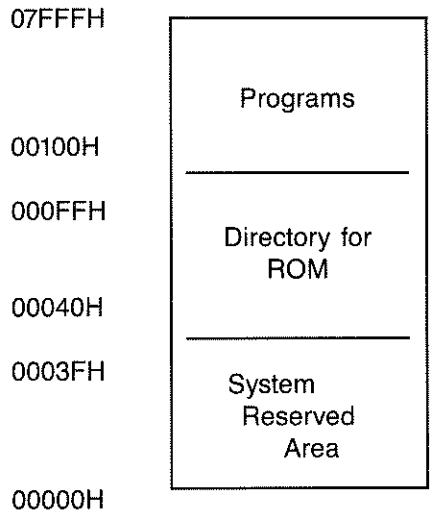
There is no limit to the number of the files in any one ROM. However, there is a practical limit to the total of the files in all the ROMs due to the menu's display format. Under no circumstances should the total files in all ROMs (i.e. ROMs #0, #1, #2, and the ROM Cartridge or I/O ROM Cartridge) exceed 65. This limit is imposed because, if a default micro floppy drive disk contains 128 files and there exists more than 65 files in the ROMs, a garbaged Menu display will result. (128 is the maximum number of files on a micro floppy.)

SECTION 3

MAPPED ROM

Contents of the Mapped ROM

The bank-switching function of the menu program causes the currently selected Mapped ROM to be physically addressable by the CPU, at addresses 00000H to 07FFFH. Therefore, any programs in a Mapped ROM must be completely contained within that single Mapped ROM. Before performing the actual bank-switching, the menu's startup routine sets up the CP/M page zero information at memory locations 08000H to 080FFH instead of at the usual area from 00000H to 000FFH. The locations from 00000H to 000FFH in the Mapped ROM are reserved by the PC-8400's CCP and BIOS. The first 64 (decimal) bytes are used for the System Interrupt Table, the BIOS and BDOS Jump Tables, the IOBYTE and other system information. The following 192 bytes are used to store the Directory of the Mapped ROM in a special format to be discussed:



The contents of the system area should be exactly as listed below. This data is the same as it is in ROM #0 (the system ROM) except for the first 3 bytes:

Location (Hexadecimal)	Contents
00000H-00002H	Jump instruction to the warm start entry
00003H-00003H	Intel standard IOBYTE. In the PC-8400, it should be 0.
00005H-00007H	Jump instruction to BDOS
00008H-0003FH	Interrupt locations 1 through 7.

Directory in the Mapped ROM

The Directory for the Mapped ROM consists of the file names stored on the ROM and their start addresses. Only .COM type files (CP/M executable files) are to be stored here. It is therefore unnecessary to store the extension name, ".COM", after the filename. The system assumes it automatically. Instead, it is suggested that the extension be filled with 3 blank characters. In this way, the ROM fields can be distinguished from the RAM-disk and micro floppy disk files, which will have non-blank extensions. Each filename in ROM is 14 bytes long. The end of the Directory is indicated by an empty file name (null).

Directory in the Mapped ROM

11 bytes	Left justified filename field. (Unused bytes padded with spaces)
1 byte	Separator (A null (0) should be inserted between the name and the start address)
2 bytes	Start address (Low - High order)

Example Contents of Directory Field in ROM

```
Address                                        Contents
   0040  57  53  20  20  20  20  20  20  20  20  20  00  00  01
         W   S                                  .   .   .
         (File name "WS", start address is 00100H)

   004E  43  41  4C  43  20  20  20  20  20  20  20  00  00  30
         C   A   L   C                          .   .   0
         (File name "CALC", start address is 03000H)

   005B  00  00  00  00  00  00  00  00  00  00  00  00  00  00
         .   .   .   .   .   .   .   .   .   .   .   .   .   .
         ^ Null          (End of directory)
```

An example dump of the entire page zero area of a typical ROM appears at the end of this section.

Execution of a Mapped ROM Program

Selection of the Mapped ROM filename is made by positioning the block cursor upon the directory display of the chosen filename, while in the Menu mode. The return key is then pressed, causing the filename to appear (as if typed in) after the CP/M prompt. An argument to the filename (additional parameters) may be typed at this point. A subsequent press of the return key will cause the execution of the selected program, by changing the memory configuration and jumping to the address specified in the directory in the ROM.

When a Mapped ROM is executed, the CP/M Page Zero area, which would under a normal CP/M implementation be in Read/Write Memory, is in Read Only Memory. The variable information, which would normally be in Page Zero, is therefore transferred to the Read/Write Memory at locations 08000H to 080FFH, for use by the application program.

Location (Hexadecimal)	Contents
08003H	IOBYTE
08004H	Current default drive number (0 = A, 1 = B,...)
0805CH-0807CH	Default file control block produced for a transient program by the CCP
0807DH-0807FH	Optional default random record position
08080H-080FFH	Default 128-byte disk buffer (also filled with the command line when a transient is loaded under the CCP.

Returning to menu

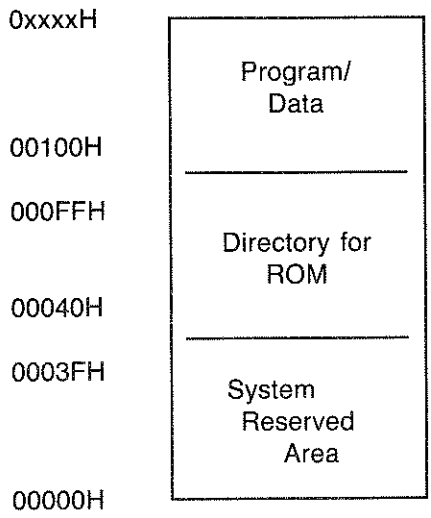
A Mapped ROM program must not use a RET command (C9H) to return to the CP/M Menu. Use JMP 0 (C3H 00H 00H, Jump Warm Boot) instead. If RET is used, the system might hang up.

SECTION 4

I/O ROM

Contents of the I/O ROM

The I/O ROM is used to store programs and data. It's memory map is identical to that of the Mapped ROM, with the exception of the size of the ROM. This can be up to 256K Bytes, and consists of the System Reserved Area, the Directory area and program/data area:



System Reserved Area

The 64 (decimal) bytes of System Reserved Area must be stored at locations 0000H to 0003FH in the I/O ROM. The contents are identical to those used by the Mapped ROM and were described earlier.

If an I/O ROM does not have this System Reserved Area filled properly, the PC-8400 will ignore the I/O ROM.

Directory Area

The contents of the directory area is identical to that of the Mapped ROM (refer to the previous description in this section of the manual, and the example dump at the end of this section.)

Execution of an I/O ROM Program

Execution of an I/O ROM program is unlike execution of a Mapped ROM program. The Mapped ROM, as previously described, is switched by the Menu program memory management into the first 32K of CPU addressable memory. The I/O ROM program, however, does not become part of the addressable memory. It is more analogous to execution of a program from the RAM-disk or the micro floppy disk in that the data is loaded from the I/O ROM into the CPU addressable RAM. This means that whichever Memory Mode (either 32K or 64K) is currently selected, determines the maximum size of CP/M program loadable from the I/O ROM into the TPA. For example, a 34K program which will load and execute properly in the 64K mode, will cause a load error if execution is attempted in the 32K mode.

The major difference in the loading of a micro floppy disk program and an I/O ROM program is that the entire micro floppy disk program is loaded into the RAM memory for execution, while only the first 256 bytes of the I/O ROM program is loaded and executed when a program in an I/O ROM is selected. The I/O ROM can be thought of as a ROM-Disk for all practical purposes. If the size of the I/O ROM program is larger than 256 bytes, the remainder of the program must be loaded by code within the first 256 bytes. (The first 256 bytes would be boot-code or an Initial Program Loader for the rest of the I/O ROM program.) This is easily accomplished by using the interbank access routine ReadIO at address 0F753H. BC enters containing the number of bytes to be read in, while A, H and L enter containing the I/O ROM address to be read. The address in the I/O ROM is passed with the most significant byte in A, the middle significant byte in H, and the least significant byte in L. DE enters containing the destination address in RAM to store data. Note that BC should not request to read in more code than the amount of TPA space available, otherwise the CP/M area above the TPA will be over-written.

The following gives a hexadecimal dump of typical page zero information for both the Mapped and I/O ROMs:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	C3	03	F6	FF	FF	C3	06	E8	C3	44	F6	FF	FF	FF	FF	FF
10	C3	69	F6	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
20	FF	FF	FF	FF	FF	FF	FF	FF	C3	8D	F6	FF	FF	FF	FF	FF
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
40	57	53	20	20	20	20	20	20	20	20	20	00	00	01	FF	FF
50	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
60	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
90	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

CHAPTER 15

User's Guide for Interbank Access Routines

NEC Corporation

The PC-8401A supports seven interbank access routines to read and write data placed in another bank, and to execute a program in another bank. In addition, a 128 byte-long buffer is allocated in the common area as an intermediate buffer for interbank memory access.

Read a byte

Name Getbyte

Address 0F6F7H

Entry HL = Address of a byte to read
C = MMR value to swap the byte into CPU address space.

Exit A = Value of the byte.

Alters A, Flags, A' and Flags

This routine reads a byte in a bank which is outside of the CPU address space. On entry, the C register holds the Memory Mapping Register value by which the byte is swapped into the CPU address space (see the chapter on Memory in this manual.) The HL register pair holds the address of the byte to read.

This routine enables interrupts.

Write a byte

Name Putbyte

Address 0F70BH

Entry A = Value to write
HL = Address where the byte is written
C = MMR value to swap the byte into CPU address space.

Exit None

Alters A, Flags, A' and Flags

The memory to be mapped into the CPU address space (passed in C) is sent to the Memory Mapping Register. This routine then writes the given value (in A) into the specified memory location (addressed by register pair HL.)

This routine enables interrupts.

Read a string

Name Getstr

Address 0F71FH

Entry B = Length of string to read
HL = Address of the string
C = MMR value to swap the string into address space.
DE = Address in common area where the string is passed.

Exit None

Alters All

This routine reads a string of given length that is in a memory area located outside of the current CPU address space.

The DE register pair holds the address in the common memory area into which the string is returned. (Usually, this area is the block/deblock buffer or the Combuf. See Combuf, below.)

The C register holds the Memory Mapping Register value to the map the memory containing the string into the CPU addressing space. The HL register pair holds the address of the string in the CPU current memory.

If the register value of B is zero upon entry, nothing is done.

Interrupts are enabled upon exit.

Write a string

Name Putstr

Address 0F71FH

Entry B = Length of string to write
HL = Address of the string in common area
DE = Address in user memory where the string is stored
C = MMR value to swap the user memory into address space

Exit None

Alters All

This routine writes a string located in the common memory area into memory that is outside of the current CPU addressing space. Upon entry, HL holds the address of the string in the common area that is to be written, DE points to the destination address of the transfer, and C holds the Memory Mapping Register value which switches the destination memory into the CPU addressing space. The B register holds the length of the string, which must be greater than 0.

Interrupts are enabled upon exit.

Transfer control to a program in another bank

Name Execfar

Address 0F733H

Entry HL = Address of the program to execute
C = MMR value to swap the program into address space

Exit None. This does not return to caller.

Alters A, flags, A' and flags'.

Example:

```
        DI                  ;No interrupts.
        LD SP, HISSTK       ;Setup stack for the program to
                            ;execute. Note that this stack
                            ;must be available after the MMR
                            ;value causes the memory switch.
        LD HL, ENTRY        ;Entry address of the program.
        LD C,HISMMR         ;MMR value that swaps the program in.
        JP EXECFAR          ;Go to the program.
```

The Execfar is used to transfer control to the routine which is outside of the current CPU addressing space. Interrupts are enabled after the Memory Mapping Register value is changed, and control is given to the target routine. Take extreme care to set up the correct Stack Pointer location before performing the jump to Execfar. The stack must be set to RAM which is mapped into the CPU addressing space when the given value is sent to the MMR. Usually, the stack to be used is outside the addressing space of the program which issues the Execfar, so the interrupts must be disabled prior to the jump to Execfar. The Execfar routine itself does not use the stack.

Execute a subroutine in another bank

Name Callfar

Address 0F738H

Entry C = MMR value that swaps the subroutine into CPU space.
HL = Address of subroutine outside of current CPU space.
DE = Return address to the current (entry) CPU space.

Exit Values returned by the routine.

Alters A

Example:

```
        DI                  ;No interrupts.
        LD SP, HISSTK       ;This stack must be available after
                            ;the MMR causes the memory switch.
        LD HL, ENTRY        ;Address of subroutine to call.
        LD DE,MYRET         ;Where control comes back.
        LD C,HISMMR         ;MMR to swap the routine in.
        JP CALLFAR          ;Don't use "CALL" here.


MYRET:
        etc.                ;Control returns here.
```

Callfar is similar to Execfar, except Callfar is used to invoke a subroutine. The RET instruction at the end of the far-called subroutine returns control to the address specified by DE in the original memory mode (that is, the memory mode which jumped to Callfar). As in Execfar, the stack memory that is pointed to by the Stack Pointer must be useable by the target subroutine after the Memory Mapping Register switches the far memory in to perform the subroutine. Interrupts must be disabled prior to jumping to Callfar.

The Callfar is invoked by a JMP rather than a CALL.

Read I/O ROM

Name Readio

Address 0F753H

Entry BC = Number of bytes to read
A:HL = Address to read in I/O ROM
DE = Destination Address

Exit None

Alters All

This routine is used to read data or programs in the I/O mapped ROM into the current CPU addressing space. Up to 256 Kbytes can be stored in the I/O mapped ROM. On entry, A holds the high two bits of the address in the I/O mapped ROM. HL holds the low 16 bits. The DE register pair points to the destination in the current CPU memory area, where the data from the I/O ROM is to be stored. The BC register pair holds the number of bytes to be read.

Buffer in common area

Name Combuf

Address 0F7D7H

Length 128 bytes

The PC-8401A provides a 128 byte long buffer in the common area for public use. The buffer, which is called Combuf, can be used as an intermediate buffer for Getstr and Putstr.

Note that the BIOS uses this buffer. When an application program needs this buffer to get or put a string utilizing Getstr or Putstr, the Combuf must be saved by copying it into the application program's own buffer. Before any BIOS or BDOS call is made, the contents of the saved copy of Combuf must be restored.

CHAPTER 16

BDOS Patches for the PC-8401A

The patches made to the standard BDOS can be categorized into the following 2 groups:

1. JUMP instructions to reboot.

   The original BDOS uses a JMP 00000H to reboot. In the PC-8401A BDOS implementation, all jumps go to a tiny routine in the common area which is at address 00000H when in the RAM user mode, or at the BIOS Warm Boot entry (0F603H) when in the ROM mode. This is because in the ROM user mode, address 00000H contains a jump to the system initializer code rather than to the BIOS Warm Boot entry.

   Such jumps could always come directly to the BIOS Warm Boot entry rather than by passing through address 00000H; however, this always terminates XSUB, and might cause a loss of compatibility with applicaton programs that alter the jump at address 00000H to trap BDOS errors and Control-C (^C).

2. Instructions to read and write the I/O byte.

   The original BDOS assumes that the I/O byte is always placed at address 00003H. The PC-8401A locates the I/O Byte at address 08003H only when a ROM program is executed.

The actual patches are listed below. All addresses are relative to the BDOS base address.

I/O byte patches

(patches are given in 8080 op. codes)

```
02ED        call    getIOb      ;Get I/O byte.
02F3        lhld    IOloc
```

GetIOB routine

The GetIOB returns the I/O byte value in A. The word IOLOC is setup by the CCP to 00003H when executing a RAM program, or to 08003H when executing a ROM program.

The GetIOB routine is placed in the common area.

```
getIOb::
        lhld    IOloc       ;Get address of I/O byte location
                            ;in this CP/M mode.
        mov     a,m         ;And get I/O byte value there.
        ret
```

Reboot patches

All the jump instructions in the list below now go to the BIOS reboot routine in the common area.

00A1	jz	reboot
00B7	jmp	reboot
013D	jz	reboot
02BA	jz	reboot

Reboot routine in Common area

```
;Comes here from BDOS when reboot took place.

reboot::

        in      mmr             ;Who called BDOS ?
        ani     00001100B       Pickup rom/ram bits.
        jz      wboot           ;ROM program called. Go directly
                                ;to Warm Boot entry.
        jmp     0               ;Otherwise jump to 0. This is to
                                ;avoid ^C or BDOS error
                                ;terminating XSUB.
```

CHAPTER 17

Function Code Definitions of the PC-8401A BIOS

SECTION 1

SCREEN DRIVER FUNCTIONS

Name	*Code*	*Function*
Chput	000H	Output a character to console.
Litout	001H	Output a character literally.
Bitout	002H	Output a character with given font.
Selcon	003H	Select console.
Askcon	004H	Return current console.
Setatr	005H	Set attributes.
Askatr	006H	Return current attributes.
Revchr	007H	Reverse characters.
Setmode	008H	Set screen modes.
Askmode	009H	Return current screen mode.
Csrctl	00AH	Enable/Disable cursor display.
Askcsr	00BH	Return current cursor display status.
Locate	00CH	Locate cursor.
Goup	00DH	Move cursor up.
Godown	00EH	Move cursor down.
Goright	00FH	Move cursor right.
Goleft	010H	Mover cursor left.
Index	011H	Index.
Rindex	012H	Reverse index.
Askpos	013H	Read current cursor position.
Setscr	014H	Set Scroll Region.
Askscr	015H	Return current scrolling region.
Eradsp	016H	Erase display.
Eralin	017H	Erase line.
Selset0	05CH	Select G0 character set.
Selset1	01CH	Select G1 character set.
Askset	01DH	Return current character set.
Tabset	01EH	Set a tab stop at cursor position.
Restab	01FH	Clear one or all tab stops.
Resscn	020H	Reset screen.
Readpix	021H	Read pixels in a line.
Wrtpix	022H	Write pixels in a line.

SECTION 2

GRAPHICS DRIVER FUNCTIONS

Name	*Code*	*Function*
Move	025H	Move graphics cursor position.
Pset	026H	Plot a pixel.
Line	027H	Draw a line.
Box	028H	Draw a box.
Boxfill	029H	Draw a box and fill inside.
Setcol	02AH	Set graphics color (1 or 0.)
Askgcp	02BH	Interrogate graphics cursor position.
Askcol	02CH	Interrogate current graphics color.
Point	02DH	Return value of pixel.

SECTION 3

COMM DRIVER FUNCTIONS

Name	*Code*	*Function*
Initcom	02EH	Initialize comm channel.
Clscom	02FH	De-activate comm channel.
Rcvcom	030H	Receive a character from comm channel.
Sendcom	031H	Send a character to comm channel.
Pollcom	032H	Sense comm channel.
Xonxoff	033H	Set/Reset Xon/Xoff flow control.
Sendbrk	034H	Send break signal.
Carrier	035H	Sense carrier status.

SECTION 4

CASSETTE DRIVER FUNCTIONS

Name	*Code*	*Function*
Initcas	036H	Initialize cassette interface.
Clscas	037H	De-activate cassette interface.
Scrfile	038H	Search for file.
Skipfile	039H	Skip a file.
Makefile	03AH	Make a file header.
Readblk	03BH	Read a data block.
Vfyblk	03CH	Verify a data block.
Wrtblk	03DH	Write a data block.
Wrteof	03EH	Write EOF block.
Motorctl	03FH	Motor control.

SECTION 5

TOD CLOCK DRIVER FUNCTIONS

Name	*Code*	*Function*
Settime	040H	Set time to TOD clock.
Gettime	041H	Get current time.

SECTION 6

FLOPPY RELATED FUNCTIONS

Name	*Code*	*Function*
Fread	042H	Read sectors.
Fwrite	043H	Write sectors.
Ffrmt	044H	Format floppy.

SECTION 7

KEYBOARD RELATED FUNCTIONS

Name	*Code*	*Function*
Kbsts	045H	Sense KB status.
Kbin	046H	Read a character from KB buffer.
Kbflsh	047H	Flush KB buffer.
Setsftky	048H	Set softkey string.
Expctl	049H	Set mode of softkeys expansion.
Kbufsts	04BH	Sense number of characters in KB buffer.
Lookchr	04CH	Scan KB buffer for particular char.
Defcsr	04DH	Define function key.
Csrmod	04EH	Select cursor mode.
Askcmod	056H	Ask cursor key mode.
Repcnt	05BH	Set key repeat interval.

SECTION 8

RAM DISK RELATED FUNCTIONS

Name	*Code*	*Function*
Rclrdsk	04FH	Initialize RAM disk.

SECTION 9

MODEM RELATED FUNCTIONS

Name	*Code*	*Function*
Dsetup	050H	Setup for dialing.
Dial	051H	Dial a digit.
Wait	052H	Wait.
Selline	053H	Select line connection.
Modmode	054H	Setup Modem mode.
Modsts	055H	Ask Modem mode.

SECTION 10

SLEEP/WAKE UP RELATED FUNCTIONS

Name	*Code*	*Function*
Autpoff	057H	Setup time to Automatic Power Down.
Slepmod	058H	Enter Sleep Mode.
Waktim	059H	Setup wakeup time.
Askwktim	05AH	Return current wakeup time setting.