PCW Hardware

John Elliott

21st February 2004

Abstract

This document describes what I have found out about the hardware of the Amstrad PCW, in the course of writing JOYCE. Some of this has been deduced by me; a lot has come from other sources.

Contents

1 General principles **4**

2 Memory **4**
2.1 PCW (“extended”) paging mode 4
2.2 CPC (“standard”) paging mode 4

3 The screen **5**
3.1 I/O ports 5
3.2 The Roller-RAM 6

4 Interrupts **6**
4.1 Timer interrupts 6
4.2 Floppy controller interrupts 6
4.3 Serial port interrupts 6
4.4 Daisywheel interrupts 6

5 Printer ports **7**
5.1 PCW8256/8512/9256/10 printer controller 7
5.1.1 I/O ports 7
5.1.2 Printer commands 7
5.1.3 Example command sequences 9
5.1.4 PCW matrix printer fonts 9
5.2 PCW9512 printer controller 9
5.3 PCW9512 PAR port 11
5.4 CPS8256 CEN port 12
5.5 Standalone CEN port 12

6 Other interfaces **12**
6.1 CPS8256 12
6.1.1 Z80-DART registers 12
6.1.2 CPS8256 Centronics port 13
6.1.3 CPS8256 Serial port 13
6.2 SCA Mark 2 Interface 14
6.3 LocoLink interface 15
6.4 Prototype serial interface 16

7 Floppy drives **16**
7.1 The floppy controller and other ports 16
7.2 Floppy drive support 16
7.3 Floppy controller probe 17
7.3.1 Detect floppy controller type 17
7.3.2 Detect drive type 17

8 Hard drives **17**
8.1 Cirtech Gem 18

9 Keyboard **18**
9.1 Keyboard Joystick(s) 18

10 Pointing Devices **19**
10.1 AMX Mouse . . . 19
10.2 Kempston Mouse . . . 19
10.3 Keymouse . . . 20
10.4 Electric Studio light pen . . . 20

11 Joysticks **20**
11.1 Keyboard joystick(s) . . . 20
11.2 Kempston interface . . . 20
11.3 Spectravideo interface . . . 20
11.4 Cascade interface . . . 21
11.5 DKTronics interface . . . 21

A Dot matrix printer fonts **22**
A.1 The character width table. . . . 22
A.2 The NLQ font . . . 22
A.3 The draft font . . . 23
A.4 Some worked examples . . . 23
A.4.1 NLQ: Character 0 (à) using BIOS 1.15 . . . 23
A.4.2 Draft: Character 0 (à) using BIOS 1.15 . . . 25
A.4.3 Repetition of columns: Character 62 ('=') using BIOS 1.4 . . . 25
A.5 Matrix fonts in LocoScript 1.20 . . . 26
A.5.1 PS widths table . . . 26
A.5.2 NLQ font . . . 26
A.5.3 Draft font . . . 26
A.5.4 Z80 code . . . 27
A.6 Matrix fonts in LocoScript v1.31 . . . 27
A.7 Matrix fonts in LocoScript v1.40 . . . 27

B The LocoLink wire protocol **28**
B.1 Basic concepts . . . 28
B.1.1 Bit mapping at the PC end . . . 28
B.2 Link idle . . . 28
B.3 Sending . . . 28
B.3.1 Sending a byte . . . 28
B.3.2 Sending a packet . . . 29
B.4 Receiving . . . 29
B.4.1 Receiving a byte . . . 29
B.4.2 Receiving a packet . . . 29
B.5 Example . . . 30
B.6 Startup sequence . . . 30
B.7 While the link is running . . . 31

1 General principles

The PCW's I/O is (for the most part) conducted using the Z80's IN and OUT instructions, rather than memory-mapped I/O. In nearly all cases, only the bottom 8 bits of the port number are decoded, and so ports are specified with 2-digit identifiers (eg: port F0h, not port 00F0h).

2 Memory

The PCW memory is divided into blocks of 16k. A PCW8256 has 16 blocks, while a fully-upgraded 2Mb PCW has 128. The processor can only address 64k at a time; the I/O ports F0h-F3h are used to select which blocks it sees where.

F0h controls what the processor sees at 0000h-3FFFh.

F1h controls what the processor sees at 4000h-7FFFh.

F2h controls what the processor sees at 8000h-BFFFh.

F3h controls what the processor sees at C000h-FFFFh.

2.1 PCW ("extended") paging mode

Usually, values written to the memory management ports have bit 7 set and the other 7 bits set to the block number:

Bits							
7	6	5	4	3	2	1	0
1	block number						

- for example, block 4 could be selected into memory at 8000h by an OUT (0F2h),84h. If the block number is out of range then the top bits will be ignored - so attempting to select block 24 on a 256k computer would actually select block 8.

2.2 CPC ("standard") paging mode

This paging mode is not used by any CP/M or LocoScript software except perhaps the memory tester (RAMTEST.COM). It is present because the PCW was based on a never-built design for ANT[1], a successor to the CPC range.

If CPC paging mode is used, then two blocks can be in one slot at once; memory writes go to one, and reads to the other.

Bits							
7	6	5	4	3	2	1	0
0	block to read			unused	block to write		

This method only allows access to the first 128k of memory, which is another good reason why no PCW programs use it.

There is an additional port used in CPC paging: port F4h (output). The value written to this is interpreted as follows:

[1] Arnold Number Two

Bits							
7	6	5	4	3	2	1	0
lock C000..FFFF	lock 8000..BFFF	lock 4000..7FFF	lock 0..3FFF	unused			

If a memory range is set as “locked”, then the “block to read” bits are ignored; memory is read from the “block to write”.

3 The screen

The screen is a 720x256 array of pixels, each pixel twice as high as it is wide. The video controller can fetch the screen data from anywhere in the bottom 128k of RAM - see port F5h below.

3.1 I/O ports

The following ports are used by the video controller:

F5h (output) sets the address of the “Roller-RAM” within the bottom 128k of memory. The “Roller-RAM” is a 512-byte table whose format is given in section 3.2. The value written to this port can be interpreted as follows:

Bits							
7	6	5	4	3	2	1	0
block			offset / 512				

- e.g, to move the Roller-RAM to offset 3200h in memory block 4, write the value 10011001 binary (99h) to port F5h.

F6h (output) sets the vertical origin of the screen. The value written to it specifies which line of the Roller-RAM corresponds to the top line of the screen. So OUT 0F6h, 8 means that the video controller will display pixel line 8 at the top of the screen - essentially, scrolling up by 8 pixels.

F7h (output) Bits 6 and 7 of this port are used. If bit 7 is set, then the screen will display in inverse video (black on green/white). If bit 6 is set, then the screen is displayed; otherwise it will be blank.

F8h (output) This port is used for various purposes, but the two which are video-related are:

OUT F8h, 8 Disable the video controller. External hardware (a TV modulator?) must drive the screen.

OUT F8h, 7 Enable the video controller.

F8h (input) The bits that concern the video controller here are:

bit 6 Frame flyback; this is set while the screen is not being drawn.

bit 4 60Hz PCW; only the top 200 lines of the screen will be visible.

3.2 The Roller-RAM

The Roller-RAM is treated as an array of 256 little-endian words. Each word is a coded pointer to the screen bitmap for this line, formed:

Bits															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
block			offset / 16										offset		

The top 3 bits give the memory block number. Bits 0-12 give the offset of the line within the block. However, it would take 14 bits to span a 16k block, and there are only 13 left. So to convert bits 0-12 into the address of a screen line, you have to use a formula such as

```
address = (offset & 7) + 2 * (offset & 0x1FF8) ;
```

Once you have the address of a screen line, it consists of 90 bytes, at intervals 8 apart (so a line would be stored in bytes 0,8,16,24,32...). This allows 8 lines to be interleaved for ease of printing text.

4 Interrupts

Various peripherals can generate interrupts. When checking the source of an interrupt, PCW CP/M checks in the order FDC, timer, other peripherals.

4.1 Timer interrupts

The timer interrupts 300 times a second. It also increments a counter which can be read from the bottom 4 bits of port F4h. The counter goes from 0-15, and stays at 15 having got there. Reading the counter resets it.

This allows PCW system software to compensate for missed timer interrupts. On interrupt, it reads the counter, and if it is more than 1 then it knows some interrupts have been missed. If it is 0, then the timer did not interrupt; the interrupt must be from a different source.

4.2 Floppy controller interrupts

The floppy controller can be programmed to send a normal interrupt or a non-maskable interrupt (NMI) on completing a command. See Section 7.1.

If the floppy controller has tried to interrupt (regardless of whether it is set to produce an NMI, a normal interrupt, or nothing at all) then bit 5 of the value read from port F8h will be 1. Otherwise it will be 0.

4.3 Serial port interrupts

The CPS8256 interface (see section 6.1) can be programmed to generate interrupts.

4.4 Daisywheel interrupts

The daisywheel printer (see section 5.2) can be programmed to interrupt when it has finished a command.

5 Printer ports

5.1 PCW8256/8512/9256/10 printer controller

5.1.1 I/O ports

The PCW uses two I/O ports to communicate with the printer controller - ports 0FCh and 0FDh. 0FCh appears to be used to initialise the controller, while 0FDh controls the printer itself.

FCh (input) returns a controller error number. The PCW XBIOS only reads this when the controller reports an error (see bit 0 of port FDh).. Values that can be returned are:

0 Underrun

1 Printer RAM fault

3 Bad command

5 Print error

0F8h Normal operation (no error).

If this port returns any other value, then “No printer” is displayed as an error.

FCh (output) is used to send commands to the printer controller. Each command is a multiple of 2 bytes long. The meaning of the commands sent to ports FCh and FDh appears to be the same, but commands are only written to port FCh while the printer is being reset.

FDh (input) returns the status of the printer. The meanings of the bits are as follows:

Bit	Meaning
7	Bail bar - 1 if it’s in, 0 if out
6	If 0, printer is executing command. If 1, printer has finished.
5	Always 0 (for daisywheel PCWs, it’s 1 when the PCW is booted, so this bit can be used to test the printer type).
4	0 if print head is at left margin; else 1.
3	Sheet feeder present?
2	Paper sensor -1 if paper is present, else 0.
1	If 0, ready - commands can be sent to port FDh. If 1, busy - they can’t.
0	If 1, controller fault - see IN 0FCh.

FDh (output) is used to send commands to the printer. All printer commands are sent to this port except during a reset.

5.1.2 Printer commands

Printer commands are a multiple of 2 bytes long. Commands which are longer than two bytes end with the two bytes C0,00. Parameters use a vertical resolution of $\frac{1}{360}$in; the horizontal resolution depends on the speed the motor is running at, but is either $\frac{1}{720}$in or $\frac{1}{1440}$in. In the following descriptions, the unit of horizontal measure is called the “tick”.

00,00 First initialisation command sent to port 0FCh during boot up; possibly does a self-test.

A4,nn Line feed by $\frac{nn}{360}$inches. Used only for feeds of $\frac{1}{30}$in and less (ie, n <= 12). Not followed by a C0,00 command.

A8,nn;A9,nn;AA,nn;AB,nn Move the print head and print a line. Bits 0 and 1 of the command byte give the print head speed and direction:

bit 0 If set, print head moves to the right; else left.

bit 1 If set, print head moves at half speed. If not, full speed.

The second byte of the command says by how many ticks the head should move in the direction of travel before the first output is printed. The PCW matrix driver assumes that the head will actually move by 9 fewer ticks; this may be because the motor can't start instantly.

This command is then followed by a number of data commands. These are of two kinds - to print, or to move the head by a specified number of ticks.

The command to print a column is treated as a 16-bit big-endian word with the following bitfields:

- Bits 15-12 are 0.
- Bits 11-9 give the column spacing. 0 is 5 ticks, 1 is 6 ticks, ..., 7 is 12 ticks.
- Bits 8-0 are 1 to fire the pins. Bit 8 is the bottom pin, bit 0 is the top.

The other type of command moves the head. These are normally at the end (to put the print head in the right place for the next row). The bit pattern for this type of command is:

- Bits 15-13 are 1,0,0 (so this command is between 80h and 9Fh).
- Bits 12-0 are the number. If the low 8 bits are 0, add 256. This means the commands, in increasing numerical order, are: 80,01 80,02 ... 80,FF, 80,00, 81,01 etc.

Finally, the command ends:

C0,00 End of command. This moves the head a further 11 ticks in the direction of travel, because the motor can't stop instantly either.

AC,nn Line feed by $\frac{n}{360}$ inches. If nn is 0, feed $\frac{256}{360}$inches. To feed more than that, pass further commands where bits 15-13 are 1,0,0 and bits 12-0 are the encoded number to feed - eg: AC,30 92,00 C0,00

This command is also terminated by a C0,00 end sequence.

B8,00 Reset printer. Moves the print head to the left margin.

C0,00 end of command sequence.

5.1.3 Example command sequences

- 00,00 A4,01 { A9,C0 } B8,00 - sent to port 0FCh as initialisation sequence. The A9,C0 pair is sent if the print head is against the left margin after the initial reset; this ensures that wherever the head was, its ability to move has been tested.
- C0,00 B8,00 - sent to port 0FCh as printer reset sequence.
- AC,3D C0,00 - line feed at 6 lines/inch
- AB,86,0E,data,0E,data,...,0E,data, 02,00, ... (21 times) ..., 80, 04, C0,00 - print a line of graphics in the ESC L graphics mode. The AB,86 starts printing and moves the head 134 - 9 = 125 ticks to the right. The commands starting 0E are the graphics columns, spaced 12 ticks apart. The 21 02,00 commands and the 80,04 move the head 130 ticks to the right of the last column, and the C0,00 moves it a further 11 ticks. Combined with the lead-in on the next line, this leaves the head ready to print under the first unprinted column of this line.

5.1.4 PCW matrix printer fonts

Although it is not strictly part of the hardware specification, the memory layout used by the dot-matrix printer fonts under CP/M and LocoScript 1 is described in Appendix **??**.

5.2 PCW9512 printer controller

Unlike most of the other ports, the 9512 printer controller decodes its I/O address as more than just 8 bits, and is thus accessed at I/O addresses 00FCh, 01FCh and 00FDh.

The bits returned by IN (0FDh) are:

Bit	Meaning
7	1 if the printer can accept commands 0 if commands can't be sent to it
6	Usually 0. The interrupt handler checks this bit,. but the significance of it is unclear.
5	1 if the printer is idle, 0 if it's executing a command. On dot-matrix PCWs, this bit is always 0.
4	Always seems to be 1
3	Unknown
2	1 if the printer has caused an interrupt.
1	0 if bytes can be sent to the controller, else 1.
0	1 if bytes can be read from the controller, else 0.

The controller is accessed by sending two-byte commands. The first byte specifies the command, and the second contains any required data. To send a command:

1. Wait until IN (00FDh) bit 1 is zero.
2. OUT (01FCh),command_byte
3. OUT (00FCh),data_byte

If the command returns data, then to read back the data, do:

1. Wait until IN (00FDh) bit 0 is 1.
2. Read data from IN (00FCh).

To read the status of the printer, do an IN (01FCh) twice in succession. If the values read don't match, repeat until they do. The bits in the byte returned are:

Bit	Meaning if set
7	Printer failed
6	No printer
5	Ribbon present
4	Paper present
3	Cover down
2	Bail bar in
1	Controller failed
0	?

(The PCW9512 schematic shows printer signals for "SEL.HOM", "CAR.HOM", "SHIF" and "PSIF", but whether any of these corresponds to bit 0 (or indeed bits 7, 6 and 1) of this port is not known).

Daisywheel commands should be treated as big-endian words. The first 4 bits define the command; the remaining 12 bits are the parameter. If the first 4 bits are 0, then the next 4 bits define a different type of command and the last 8 bits are the parameter.

First byte High 4 bits	First byte Low 4 bits	Second byte	Effect	Returns
0	1	ribbon type 3 for singlestrike, else 0Ch	Prepare to output character Followed by 4xxx-7xxx	nothing
0	2	0	Get PAR port status	status
0	3	1 or 3	Unknown	nothing
0	4	value	Write byte to PAR port	nothing
0	5	0	Used to re-check paper sensor?	nothing
0	6	0	Used to re-check paper sensor?	byte
0	7	80h	Enable daisywheel interrupts	nothing
0	8	07Fh	Disable daisywheel interrupts	nothing
0	9	0	Reset interrupt flag (called from interrupt handler)	nothing
0	0Ah	22h, 24h, 25h	Unknown	byte
0	0Bh	3	Unknown	byte
1	2	0	Initialise controller. This is the first command sent.	0 if OK else error
2 3	The PCW9512 BIOS does not send these codes.			
4 5 6 7	Bits 12-9 are impression (2 for low, 5 for medium, 9 for high) Bits 0-7 are pin number on wheel.		Output specified character. Always preceded by a 01xx command.	nothing
8	Not sent by the PCW9512 BIOS			
9	Distance in $\frac{1}{120}$ths of an inch		Move left	nothing
A	unknown		Unknown	nothing
B	Distance in $\frac{1}{120}$ths of an inch		Move right	nothing
C	Distance in $\frac{1}{192}$ths of an inch		Feed paper forwards	nothing
D	0		Unknown	nothing
E	Distance in $\frac{1}{192}$ths of an inch		Feed paper backwards	nothing
F	0		Unknown	nothing

5.3 PCW9512 PAR port

The PAR port is part of the PCW9512 printer controller, and is controlled by commands 2 (get status) and 4 (write byte to PAR).

The status byte gives the values of the following Centronics lines:

Bit	Line
7	ACK
6	BUSY
5	ERROR
4	SELECT
3	PAPER OUT
2	Always 0?
1	Always 0?
0	Always 1?

So to do output, wait until bit 6 goes to 0 and then write the data using command 4.

Since the 9512 hardware supports the Centronics ACK signal, it is possible for a 9512 to act as the master computer in a LocoLink conversation - see section 6.3.

5.4 CPS8256 CEN port

The CPS8256 CEN port is at E3h.

To read its status, write 10h to port E3h. Then read port E3h; bit 5 is set if the printer is ready.

To write data, write the character to port E8h. Then send the following bytes to port E3h: 5, E8h, 5, 68h. These will toggle the STROBE line.

5.5 Standalone CEN port

The standalone CEN port is at ports 84h-87h.

The status is read from IN (84h). Bit 0 is1 if the printer is busy, 0 if it's ready.

To write a byte, write it to port 87h, then to port 85h, then to port 87h again. The two writes to port 87h toggle the STROBE line.

6 Other interfaces

6.1 CPS8256

The CPS8256 is based on a Z80-DART and an 8253 timer. Ports are as follows:

Port	Meaning
E0	DART Channel A data
E1	DART Channel A control
E2	DART Channel B data
E3	DART Channel B control
E4	8253 counter 0
E5	8253 counter 1
E7	8253 write mode word

To write a value to a DART register other than register 0, send the register number and then the value to the "control" port (E1h or E3h). To read a register, send the register number to the control port and then do an IN on that port.

To read / write DART register 0, just read or write the value to port E1h or E3h. When writing, the bottom 3 bits of the value must be 0; otherwise one of the other regsters would be selected.

6.1.1 Z80-DART registers

The Z80-DART registers are numbered 0 to 5. Many of the bits in these registers are not used by the CPS8256.

Register	Bit	Meaning when read	Meaning when written
0	0	Receive character available	Must be 0
	1	Interrupt pending[A]	Must be 0
	2	Transmit buffer empty	Must be 0
	3	DCD	Command
	4	Ring Indicator	Command
	5	CTS	Command
	6	Not used	Not used
	7	Break	Not used
1	0	All sent	Ext int enable
	1	Not used	Transmit int enable
	2	Not used	Status affects vector[B]
	3	Not used	Receive int mode
	4	Parity error	Receive int mode
	5	Receive overrun	Wait/Ready on R/T
	6	Framing error	Wait/Ready function
	7	Not used	Wait/Ready enable
2		Interrupt vector[B]	Interrupt vector[B]
3	0		Receive enable
	1-4		Must be set to 0
	5		RTS/CTS set automatically
	6-7		Receive bits (5,6,7,8)
4	0		Parity enabled
	1		Parity even
	2-3		Stop bits (1, 2, 3 for 1, 1.5, 2)
	4-5		Not used
	6-7		Clock multiplier(1,16,32,64)
5	0		Not used
	1		RTS
	2		Not used
	3		Transmit enable
	4		Send break
	5-6		Transmit bits (5,6,7,8)
	7		Not used

[A]Only on channel A
[B]Only on channel B

6.1.2 CPS8256 Centronics port

The Centronics port (described in section 5.4) returns its status in register 0 of Channel B. The STROBE signal is controlled by bit 7 of register 5 (what the DART thinks is DTR) and the BUSY signal shows up in bit 5 (CTS) of register 0.

6.1.3 CPS8256 Serial port

The serial port is connected to Channel A of the DART.

- To check if the port can output, read register 0 of Channel A.

- If you are using software handshaking, check bit 2 (Transmit buffer empty) and wait until it's nonzero.
- If you are using hardware handshaking, check bit 5 (Clear to Send); when this goes to 1, read register 1 until bit 0 (All Sent) is 1.
- Then, to output, write the correct byte to the data channel.

To read from the port in non-interrupt mode:

- If hardware handshaking is enabled, check bit 0 of register 0 (RX character available) and if it's 0, raise DTR (set bit 7 of register 5).
- Wait until bit 0 of register 0 becomes 1.
- Read the character from the DART data channel.
- If hardware handshaking is in use, drop DTR.

To set transmit baud rate, send 36h to port E7h, and two bytes encoded rate to port E4h. To set the receive baud rate, send 76h to port E7h, and the two bytes to port E5h.

The encoded rate is $\frac{125000}{baud}$ - send the high byte first, then the low byte.

6.2 SCA Mark 2 Interface

The SCA interface is programmed in the same way as the CPS8256, except that there's also a Real Time Clock fitted. The RTC is not emulated in JOYCE so this information has not been verified; use it at your own risk.

To read a byte from the RTC:

- Set channel A DTR.
- Reset channel B.
- For each bit: Unset channel A DTR, read the channel B Ring Indicator, and set channel A DTR. The bit obtained should be complemented. The first bit read is bit 7 of the byte; the last is bit 0.

To write a byte to the RTC:

- Set channel A DTR.
- For each bit, if it's a 1 then reset channel B. If it's a zero set channel B RTS. Then reset and set channel A DTR.
- Finally, unset channel A DTR, reset channel B, read a single bit from the channel B Ring Indicator, and set channel A DTR again.

To send an RTC command:

- To start the command: Unset channel A DTR, reset channel B, set channel B RTS then set channel A DTR.
- Send the command bytes using the write code above.

- Read any result bytes using the read code above. For all result bytes except the last: Reset channel B, set channel B RTS, unset channel A DTR, set channel A DTR and then reset channel B again.
- Finally, set channel B RTS, set channel A DTR, unset channel A DTR and then reset channel B.

To read the clock:

- Send the command D0h, 0.
- Send the command D1h; the next 4 bytes read will be hours, minutes, days, months (BCD).
- Send the command D0h, 5.
- The clock will return 3 bytes. The first is the year (BCD); I suspect the others are day and month again, which we ignore.

To write to the clock:

- Send the command D0h, 20h.
- Send the command D0h, hours, minutes, days, months (all BCD).

or:

- Send the command D0h, 5.
- Send the command D0h, year, day, month (BCD).

6.3 LocoLink interface

LocoLink is used to connect the PCW to the parallel port of another computer. The supplied software acts as a file server ("slave" in LocoLink terminology), so that the other computer (the "master" - normally a PC, though software also existed allowing a PCW9512 to do it, and the PCW16 is supposed to have built-in LocoLink support) could read or write files.

Although the LocoLink interface connects to the parallel port, it acts as a serial device, using only 4 wires (two each way). This is so that it can support non-bidirectional parallel ports.

On a PCW with the interface, the LocoLink appears at address 0FEh. The port behaves as follows:

Bit	Input meaning	Output meaning
0	Data 0	BUSY
1	Data 1	ACK

At the other end of the parallel cable, the two "Data" lines correspond to the first two data lines in the parallel port. BUSY and ACK correspond to the Centronics pins of the same name.

If a LocoLink interface is connected to both the expansion and parallel ports of a PCW9512, and bytes are sent to port 0FEh, the bits in the parallel port change as follows:

Value written		Value read	
Bit 1	Bit 0	Bit 7	Bit 6
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

JOYCE v2.1.0+ contains support for the “slave” end of a LocoLink connection.

The communications protocol used by the LocoLink interface is described in Appendix B.

6.4 Prototype serial interface

The original PCW specification mentions that the prototype PCW had a serial interface based on the Intersil IM6403 UART. Its data register was at address 0FEh for both input and output, and its status register at 0F9h (input only). JOYCE does not emulate this interface.

7 Floppy drives

The PCW floppy controller is the uPD765A, run in non-DMA mode. The FDC main status register is at I/O port 00h; its data register is at 01h.

7.1 The floppy controller and other ports

The System Control port (0F8h) has five commands which affect the floppy controller:

02 If the floppy controller interrupts, the Z80 gets an NMI.

03 If the floppy controller interrupts, the Z80 gets a normal interrupt. If the Z80 has disabled interrupts, the interrupt line stays high until the Z80 enables them again.

04 Floppy controller interrupts are ignored.

05 Set the floppy controller’s “Terminal count”, which aborts a data transfer.

06 Clear the floppy controller’s terminal count.

7.2 Floppy drive support

PCW operating systems support two floppy drives, connected to the controller as drives 0 and 1.

On a PCW 8256, 8512 or 9512, commands for drives 2 and 3 are sent to drives 0 and 1 (ie, the drive number is not completely decoded).

On a PCW 9256, 9512+ or 10, commands for drive 2 appear to go to drive 1, except that the drive is always “ready”, even when the motor isn’t running. I haven’t dared to enquire what happens if drive 3 is selected.

7.3 Floppy controller probe

Versions of LocoScript and CP/M written after the PCW9256 was released contain code to discover what sort of PCW is in use - 8256/8512/9512 or 9256/9512+/10. The code then detects whether 3.5” disc drives are in use.

The test is done in two stages.

7.3.1 Detect floppy controller type

1. Stop disc motors.
2. Send SENSE DRIVE STATUS for drive 0 to the disc controller until it reports drive 0 is not ready.
3. Send SENSE DRIVE STATUS for drive 2. On an 8256, 8512 or 9512, this returns the same status as drive 0 (ie: not ready). But on a 9256, 9512+ or 10, it returns a status of “ready”.

7.3.2 Detect drive type

If a 9256-style controller was found, separate checks are then done on each drive to see whether they are 3.5” or not. The test is performed once on drive 0 and once on drive 2. The test on drive 2 will return results that are valid for drive 1, for some strange reason.

1. Start disc motors.
2. Recalibrate the drive (ie, move the head to track 0).
3. Send SENSE DRIVE STATUS to the drive.
4. If the “Track 0” bit is not set:
   - For drive 0: If the “read-only” bit is set, the drive is 3”. Otherwise it is 3.5”.
   - For drive 2: The drive does not exist.
5. If the “Track 0” bit is set:
   - Turn off the disc motors, and wait for them to stop.
   - Send SENSE DRIVE STATUS to the drive again. If “Track 0” is set, the drive is 3”. Otherwise it is 3.5”.

8 Hard drives

The only hard drive whose driver I have been able to study is Cirtech’s Gem drive. It is not emulated in JOYCE so this information is not verified (there’s hardly enough of it to verify).

8.1 Cirtech Gem

The Gem drive uses three I/O ports: numbers 0A8h, 0A9h and 0AAh. These behave as follows:

0A8h Data register. When sectors are read or written, the data go to/from this port.

0A9h Possible status register.

0AAh Writing to this port may reset the controller.

9 Keyboard

The keyboard appears as a memory-mapped device at 3FF0h-3FFFh in memory block 3.

- Keys 0-71 correspond to bit (n mod 8) of byte (n / 8) of the map.
- Key 72 corresponds to bit 7 of byte 9.
- Keys 73-80 correspond to bits 0-7 of byte 10.

The keys q w e r o p [ ] a s d f g h j # [SHIFT] z x c v b n m [GRID] [SPACE] and the four cursor keys also set bits in bytes 14 and 15.
[f1] (key 2) also sets bit 1 of byte 12.
[f3] (key 0) also sets bit 0 of byte 12.
[SPACE] (key 47) also sets bit 4 of bytes 12, 14 and 15, and bit 5 of byte 13.
[SHIFT] (key 21) also sets bit 5 of bytes 14 and 15.
Byte 13 bit 7 is always set.

9.1 Keyboard Joystick(s)

The keyboard memory map on the PCW is nice and regular until the hiccup at key 72. Oddly enough, it also exactly matches the keyboard matrix schematic in the PCW9512 technical manual - until key 72.

The keyboard schematic also has entries in its matrix table for one or two joysticks. (Existing PCWs have no provision for connecting a joystick to the keyboard, but the PC1512 does). This may have been how the joystick(s) on ANT were to have been implemented.

Looking at the keyboard membrane diagram and the keyboard layout in memory, it’s possible to guess how the key numbering scheme may have looked originally:

Key	Current number	Position on current keyboard membrane	Conjectural'original' number
F3 up to Z	0 to 71	0 to 71	0 to 71
Joystick 1 up		80	72
Joystick 1 down		81	73
Joystick 1 left		82	74
Joystick 1 right		83	75
Joystick 1 fire 2		84	76
Joystick 1 fire 1		85	77
Spare		86	78
<-DEL	72	79	79
F5/ F6	73	72	80
EXTRA	74	73	81
CAN	75	74	82
[-]	76	75	83
F7 / F8	77	76	84
Enter	78	77	85
Cursor down	79	78	86
ALT	80	87	87
Joystick 2		88 to 93	88 to 93

My guess would be that when the joysticks were removed, this created seven spare spaces between Z and DEL. The membrane was then redesigned to move the keys F5 to Cursor Down into this gap, and the keyboard controller reprogrammed so that the key codes returned to the host computer were the same as before.

10 Pointing Devices

10.1 AMX Mouse

This mouse appears at ports 0A0h-0A3h:

0A0h gives vertical movement - low nibble = number of moves up, high = number of moves down

0A1h gives horizontal movement - low nibble = number of moves right, high = number of moves left

0A2h gives button state in bottom 3 bits. These are 0 if the button is pressed, else 1. Bit 0 is the left button; Bit 1 is middle; Bit 2 is right.

0A3h is written to by Stop Press. The values 93h (reset?) and 0FFh followed by 00h (zero the counters?) are used.

10.2 Kempston Mouse

This mouse appears at ports 0D0h-0D4h:

0D0h gives X position, 0-255.

0D1h gives Y position, 0-255.

0D4h gives button state in bottom 2 bits. Zero if the button is pressed, else 1. Bit 0 is left; bit 1 is right.

10.3 Keymouse

The Keymouse appears as a memory-mapped device, replacing some of the redundant keyboard mappings.

3FFBh Bits 6-0: horizontal movement counter.

3FFCh Bits 7-6: high bits of vertical movement counter.

3FFDh Bits 3-0: low bits of vertical movement counter.

3FFEh Bits 7: Left button. Bit 6: Right button.

10.4 Electric Studio light pen

The lightpen appears at ports 0A6h and 0A7h. The method it uses to encode its position is not known.

11 Joysticks

11.1 Keyboard joystick(s)

These are described in section 9.1.

11.2 Kempston interface

The Kempston interface is visible at port 9Fh. It appears to have the same bit assignments as the Spectrum version of the interface:

Bits 7-5 Ignored.

Bit 4 1 if the fire button is pressed.

Bit 3 1 if the joystick is pushed down.

Bit 2 1 if the joystick is pushed up.

Bit 1 1 if the joystick is pushed left.

Bit 0 1 if the joystick is pushed right.

11.3 Spectravideo interface

The Spectravideo joystick interface appears at 0E0h (so it cannot be used at the same time as a CPS8256 interface). The values it returns are:

Bit 7 Always 0. If nothing is present on this port, 1 is returned.

Bit 6 Ignored.

Bit 5 Ignored.

Bit 4 1 if the joystick is pushed to the right.

Bit 3 1 if the joystick is pushed up.

Bit 2 1 if the joystick is pushed to the left.

Bit 1 1 if the fire button is depressed.

Bit 0 1 if the joystick is pushed down.

11.4 Cascade interface

The Cascade joystick interface also appears at 0E0h. The 'Head over Heels' driver for this joystick doesn't work with a Spectravideo interface; it uses these bits:

Bit 7 0 if the fire button is pressed..

Bit 6 Ignored.

Bit 5 Ignored.

Bit 4 0 if the joystick is pushed up.

Bit 3 Ignored.

Bit 2 0 if the joystick is pushed down.

Bit 1 0 if the joystick is pushed to the right.

Bit 0 0 if the joystick is pushed to the left.

11.5 DKTronics interface

The DKTronics interface appears to be one register on a larger chip. To read it, write 0Eh to port 0AAh and then read port A9h. The use of register 0Eh and the presence of a "DKTronics sound" driver in Head Over Heels suggests that the chip may be an AY-3-8912 or similar sound generator.

Head Over Heels uses these bits of the value that it reads:

Bit 7 Ignored.

Bit 6 0 if the fire button is pressed..

Bit 5 0 if the joystick is pushed up.

Bit 4 0 if the joystick is pushed down.

Bit 3 0 if the joystick is pushed to the right.

Bit 2 0 if the joystick is pushed to the left.

Bit 1 Ignored.

Bit 0 Ignored.

A Dot matrix printer fonts

The PCW dot-matrix fonts are stored in three tables in Bank 2. The addresses and sizes of these tables differ from CP/M version to CP/M version; you can use LPT8FONT[2] to discover where the fonts are for a given CP/M version. Known font addresses are:

CP/M version	Font starts at
1.1, 1.2,1.4	5E28h
1.12,1.14,1.15	5EBDh

There are three consecutive tables of font data:

A.1 The character width table.

This is always 64 bytes long, and used for spacing the characters when proportional mode is selected. To get the width for character <n>, read byte (<n>/2); the high nibble will be for the even-numbered character (0,2,4,...) and the low nibble for the odd-numbered one (1,3,5,...)

A.2 The NLQ font

The font starts with a table (the character offset table) containing 129 words. Each word has the following bitwise structure:

Bit 15: The character has a descender; print it on the bottom 8 pins rather than the top 8.

Bits 14-12: Space to put at the left of this character.

Bits 11-0: Offset of the specification of this character, from the start of the font.

The last entry gives the offset of the first byte after the font. This means that for any character, the size of its description can be found by taking offset(char+1) - offset(char). In the standard fonts, offset(128) also gives the offset of the start of the draft font from the start of the NLQ font.

After the character offset table is a pattern table. Each entry in it is two bytes long; the high byte is printed on the first pass, and the low byte on the second (or vice versa? The high byte is drawn slightly above the low one, anyway). The least significant bit corresponds to the top pin.

The length of the pattern table isn't that important, but in the standard fonts it can be deduced simply by subtracting 258 from the offset of character 0.

After that we have the character descriptions. As mentioned above, for each character, its description is from offset(char) to offset(char+1). An entry is a stream of bytes. If the top bit is set, it means: Leave a blank column before printing this column. The values of the low 7 bits are:

00h-79h: Multiply by 2 to get an offset into the pattern table. Then take the two bytes at that offset for the first and second pass.

7Ah: The two bytes after this code are printed on the first and second pass.

[2] <http://www.seasip.demon.co.uk/Cpm/software/amstrad.html>

7Bh-7Fh: Repeat the next pattern (byte - 79h) times, putting a blank column before the second and subsequent repetitions.

A.3 The draft font

As with the NLQ font, the draft font starts with a character offset table, in which every word is formed:

Bit 15: The character has a descender; print it on the bottom 8 pins rather than the top 8.

Bits 14-12: Space to put at the left of this character.

Bits 11-0: Offset of the specification of this character, from the start of the font.

The last entry gives the offset of the first byte after the font. Fonts can use memory up to and including 6BFFh.

After the character offset table is a pattern table. Each entry in it is a single byte, since draft mode only prints one pass. The least significant bit corresponds to the top pin.

The length of the pattern table isn't that important, but in the standard fonts it can be deduced simply by subtracting 258 from the offset of character 0.

After that we have the character descriptions. As mentioned above, for each character, its description is from offset(char) to offset(char+1). An entry is a stream of bytes. If the top bit is set, it means: Leave a blank column before printing this column. The values of the low 7 bits are:

00h-79h: This is an offset into the pattern table. The bit pattern for the next column is the byte at that offset.

7Ah-7Fh: Repeat the next pattern (byte - 78h) times, putting a blank column before the second and subsequent repetitions.

(note that unlike the NLQ print, there is no literal bitmap type).

A.4 Some worked examples

A.4.1 NLQ: Character 0 (à) using BIOS 1.15

Looking in the NLQ table, the first 2 entries are 01F6h 0209h. The character bytes between those offsets (60F3h - 6106h) are (all values in hex):

```
2E 14 15 03 08 1A 08 7A 01 44 08 4C 08 36 20 0A 2B 02 82
```

Byte(s)	First pattern	Second pattern
2E	30	20
14	04	40
15	00	10
03	00	44
08	08	00
1A	00	45
08	08	00
7A 01 44	01	44
08	08	00
4C	00	46
08	08	00
36	20	04
20	08	40
0A	04	00
2B	38	38
02	00	40
82	(gap) 00	(gap) 40

Which gives us the following two patterns - one from the first set of bytes:

```
3000000000020030 0
0400808180808480 0
.......#..........
.                 .
.#            #   .
.   # # # # # # #   .
#               #  .
#           #   #  .
.                 .
...................
```

and one from the second:

```
2414040404004034 4
0004050406040080 0
.....#............
.         #       .
.  # # # # # #     .
.               #  .
. #             #  .
#               #  .
.# # # # #  #   # #
...................
```

If we take a row from the second pattern, then the first, and so on, then the letter appears:

```
.....2............
.      1          .
.        2        .
```

```
.                .
.  2 2 2 2 2     .
.1           1   .
.              2  .
.   1 1 1 1 1 1  .
. 2            2  .
1              1  .
2              2  .
1          1   1  .
.2 2 2 2 2   2   2 2
.                 .
.                 .
...................
```

A.4.2 Draft: Character 0 (à) using BIOS 1.15

In the draft table, the first two words are 0157h 015Ch. Adding these to the base of the font gives 6948h and 694Dh; so the bytes between 6948h and 694Ch inclusive are the pattern for character 0.

The bytes are: 05 0E 97 C7 92 01 and these expand to:

05 -> patt[5] = 0x20

0E -> patt[14] = 0x54

97 -> patt[23] = 0x55, blank column first

C7 -> patt[71] = 0x56, blank column first

92 -> patt[18] = 0x38, blank column first

01 -> patt[1] = 0x40

and, translated to a bitmap, these are:

```
25 5 5 34
04 5 6 80
...#.....
.    #  .
.# # #  .
.      #.
.# # # #.
#      #.
.# # #  #
.........
```

A.4.3 Repetition of columns: Character 62 ('=') using BIOS 1.4

This character is number 62. So 124 bytes from the start of the offset table, we find the two words 02B3h, 02B5h. Thus we know the character description fits in 2 bytes, and starts 02B3h bytes from the start of the font.

The two bytes found there are:

7Dh [repeat 5 times]

0Ah [pattern]

Entry 0Ah in the pattern table is 14h, so the character is formed:

```
. . . . . . . . .
.       .
# # # # #
.       .
# # # # #
.       .
.       .
. . . . . . . . .
```

with the blank columns inserted automatically.

A.5 Matrix fonts in LocoScript 1.20

The fonts are stored in the file MATRIX.STD on the boot disc, in the same order as in CP/M. The data start at an offset of 011Ah from the start of the file.

LocoScript 1 is able to print 224 characters - numbers 0-127 and 160-255. These are stored as a single block of bitmaps in MATRIX.STD, which subtracts 32 from character codes above 160 to create its internal character index.

The design of MATRIX.STD as a separate file seems to imply that other dot-matrix fonts could be loaded in LocoScript 1. However the only alternative LS1 fonts I've seen (in Digita International's Supertype) simply patch the existing MATRIX.STD file.

Note that later versions of LocoScript 1 (v1.30+) use the same font format, but different file formats - see below.

A.5.1 PS widths table

There are 224 characters rather than 128, so the PS widths table is 112 bytes long.

A.5.2 NLQ font

Since there are 224 characters, the offsets table is 225 words long. The offsets to characters start at 0; ie, what must be added is the address of the character description table, not the address of the font itself. The offset table is at 18Ah in the file, and the character description table is at 442h. The character description bytes are almost the same as in CP/M, but the special values of the low 7 bits are slightly different:

00h-7Ah: Multiply by 2 to get an offset into the pattern table. Then take the two bytes at that offset for the first and second pass.

7Bh: The two bytes after this code are printed on the first and second pass.

7Ch-7Fh: Repeat the next pattern (byte - 79h) times, putting a blank column before the second and subsequent repetitions.

A.5.3 Draft font

The draft font is at offset 1181h in MATRIX.STD. Its offsets table is again 225 words long; offsets are based on the font address, like in CP/M.

A.5.4 Z80 code

The first 11Ah bytes of MATRIX.STD are the Z80 code that generates font bitmaps. Again, this seems to suggest that alternative fonts were planned; the people designing the characters would have been able to use any code they liked to generate them. Very similar code is used under CP/M.

The entry point is at the beginning of MATRIX.STD, and takes the following parameters:

```
 A = character
BC = address of this routine (MA-
TRIX.STD must be prepared to be loaded any-
where in memory).
DE = address of a byte to which charac-
ter width should be written.
```

On return, the registers should be:

```
 A = character
BC = address of the code that generates the char-
acter bitmaps
PS width of character stored.
DE = 1 + entry DE.
HL corrupt.
All other registers and flags preserved.
```

The character bitmap generation code will then be called with:

```
 A = character
BC = address of this routine
DE = address of 24-
byte buffer in which to store the generated char-
acter bitmap.
H = 0FFh for NLQ, else draft.
Bit 0 of L is 0 for pass 1, 1 for pass 2 (or vice versa?)
```

On return, the registers should be:

```
BC IX corrupt.
DE incremented by number of bytes writ-
ten to the buffer.
All other registers and flags preserved.
```

A.6 Matrix fonts in LocoScript v1.31

The MATRIX.STD file in this version of LocoScript has a 128-byte header, so 80h needs to be added to all offsets.

A.7 Matrix fonts in LocoScript v1.40

This version of LocoScript doesn't have a MATRIX.STD file. Instead it has a PRINTER.JOY. The format of the data is the same, but 1D40h needs to be added to all offsets.

B The LocoLink wire protocol

The description of the LocoLink protocol is derived from examination of the “slave” program (LLINK202.EMS). The protocol appears to be symmetrical - ie, the “master” and the “slave” go through the same steps to transmit a packet. However, it’s easier to see what is going on at the “slave” end, where the LocoLink interface presents the data directly to the CPU, than at the “master” end where the parallel port interface gets slightly in the way - see section B.1.1.

The PCW16 version of LocoLink appears to use a later protocol which works slightly differently. These differences will be noted in the text.

B.1 Basic concepts

LocoLink works with two wires in each direction - each computer can control the values of two, and read the values of the other two. The values taken together form a two-bit number (0-3) and it’s most convenient to describe the protocol in these terms. For PCW - to - parallel communications, ACK is the high bit of the number and BUSY is the low bit.

B.1.1 Bit mapping at the PC end

On a PC parallel port, these lines are swapped over (BUSY appears on bit 7 and ACK on bit 6) and the sense of ACK is inverted (it’s 1 if the PCW is sending 0, and vice versa). Additionally, when output is being made, the STROBE line must be raised and then lowered; otherwise the LocoLink interface does not detect the changed values.

B.2 Link idle

When the link initially starts, the slave sends 2 and the master sends 3. In the protocol described below, this means that the slave is listening for the first packet.

B.3 Sending

B.3.1 Sending a byte

Note: Bytes must only be sent in packets - see below.

- Wait until the value sent by the other end goes from 3 to 1.
- Send 2 or 3, depending whether bit 7 of the byte is 1 or 0.
- Wait until the value sent by the other end goes from 1 to 3.
- Send 0 or 1, depending whether bit 6 of the byte is 1 or 0.
- Wait until the value sent by the other end goes from 3 to 1.
- Send 2 or 3, depending whether bit 5 of the byte is 1 or 0.
- ... and so on until all the bits of the byte have been sent.

B.3.2 Sending a packet

Before sending a packet the value sent by the other end should be 2. It may also be 3; if so, send 3 and wait for it to change to 2.

- Send 0.
- Wait until the value received goes from 2 to 3.
- Send 1.
- Send one byte: the packet type.
- Send one byte: number of following bytes (can be 0 for none).
- Send any following bytes.
- Send two bytes: the checksum (CRC?) of the packet.
- Wait until the value received goes from 3 to 1.
- Send 3.
- Wait until the value received goes from 1 to 3.
- Send 2.

B.4 Receiving

B.4.1 Receiving a byte

Note: Bytes are only received as part of packets - see below.

- Wait until the value sent by the other end becomes 2 or 3. The low bit of the value gives bit 7 of the byte being read.
- Send 3.
- Wait until the value sent becomes 0 or 1. This gives bit 6 of the byte being read.
- Send 1.
- Wait until the value sent becomes 2 or 3. This gives bit 5 of the byte being read.
- Send 3.
- ... and so on until all the bits have been read.

B.4.2 Receiving a packet

To receive a packet:

- Wait until the value sent by the other end changes from 3 to 0.
- Send 3.
- Wait until the value sent by the other end changes from 0 to 1.

- Send 1.
- Receive two bytes (see above). The first is the packet type, and the second is the number of additional bytes that follow.
- Receive the additional bytes, if any.
- Receive the 2-byte checksum/CRC.
- Wait for the value sent by the other end to go from 0 or 1 (bit 0 of the last byte) to 3.
- Send 3.
- Wait for the value sent by the other end to go from 3 to 2.

B.5 Example

Here the master sends a packet to the slave:

Master value	Slave value	Comment
3	2	Master not listening; Slave listening
0	2	Master starts sending packet
0	**3**	Slave acknowledges
1	3	Master ready to transmit bytes
1	**1**	Slave ready to receive bytes
2	1	Bit 7 of first byte is 0
2	**3**	Bit 7 acknowledged
0	3	Bit 6 of first byte is 0
0	**1**	Bit 6 acknowledged
3	1	Bit 5 of first byte is 1
3	**3**	Bit 5 acknowledged
... skip a lot more bits like this ...		
1	3	Bit 0 of last byte is 1
1	**1**	Bit 0 acknowledged
3	1	End of packet (slave deduces this from byte count)
3	**3**	End of packet acknowledged
2	3	Master listening; slave not listening

This would seem to imply that packets must strictly alternate, since at the end of a conversation the other computer is now the one listening.

B.6 Startup sequence

The first packet exchange is as follows:

Sender	Type	Length	Additional bytes		
Master	0Bh	02h - 20h	41h	31h	Master program name, optional
Slave	C3h	01h - 1Fh	0Bh	Slave program name, optional	

In both of these packets, the program name may or may not be present. The other bytes must be exactly as given, or a “link failed to start” error will occur.

The PCW16 version expects a different packet exchange:

Sender	Type	Length	Additional bytes		
Master	5Ah	02h - 20h	46h	31h	Master program name
Slave	18h	01h - 1Fh	5Ah	Slave program name, optional	

B.7 While the link is running

The master now either:

- Hangs up the link. This is done by sending a packet of type 22h to which the slave does not reply.
- Sends a command packet. This has a type of 39h, and is 1-15 bytes long. It represents a DOS function call; only a subset of calls are supported. The data bytes in the packet are:

Byte	Meaning
0	Function (AH)
1	Subfunction (AL)
2-3	BX
4-5	CX
6-7	DX

Presumably longer versions of this packet would also contain SI, DI and BP. Most function calls don't need to pass parameters and so just send a 1-byte packet. For example, call 0Eh (log in drive) doesn't bother to send a drive number, since a LocoLink slave computer provides only one disc drive.

Once the command has been sent, the slave will send back one of the following packets:

- Success: Type is 50h. Packet length is 0-8 bytes; its payload is the values for AX, BX, CX, DX (low byte first).
- Error: Type is 67h. Packet length is 2 bytes; the first is the DOS error number (see INT 21h function 59h) and the second is 0.
- Request data. Used where the corresponding DOS function would pass data using a pointer or DMA. The packet type is 7Eh, and length is 2 bytes. The first byte is 11h to ask for an FCB, 80h to ask for a filename, and the second is the maximum number of bytes to transfer. The master will reply to this with a packet of type 95h containing the requested data.
- Return data. Used where the corresponding DOS function expects data to be passed to it by a pointer or DMA. The packet type is 0ACh; the first byte may be a type code, and the remainder contains the data. The master will reply to this with a 0C3h packet, the first byte of which is 0ACh.

The slave can also send a 0ADh packet, which must be acknowledged by a 0C3h packet, the first byte of which is 0ADh.

Once the slave has sent a "success" or "error" packet (50h or 67h) the master can send another command packet or hangup packet.

Command packets supported by LocoLink PCW 2.02 are:

Function number	Meaning	Parameters in initial packet	Additional transfers
0Dh	reset	none	none
0Eh	login	none	none
10h	close disc label		
11h	get disc label	none	Slave returns disc label if found
13h	delete disc label		
16h	create disc label		
17h	rename disc label		
36h	get free space	none	none
39h	create directory		
3Ah	remove directory		
3Bh	set current directory	none	Master sends new directory
3Ch	create file		
3Dh	open file		
3Eh	close file		
3Fh	read file		
40h	write file		
41h	erase file		
42h	set file pointer		
43h	set/get attributes		
44h	ioctl		
47h	get current directory	none	Slave returns current directory
4Eh	find first file	AL = search attribute	Master sends search path. If successful slave returns find file data
4Fh	find next file	none	none
56h	rename file		
57h	set/get timestamp		
5Bh	create file		