PCW Hardware

John Elliott

March 26, 2026

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

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 Boot ROM

Most of this information is derived from Jacob Nevins' description of the boot process (with disassembly) at <http://www.chiark.greenend.org.uk/~jacobn/cpm/pcwboot.html>.

The PCW has no separate boot ROM. Instead, on startup, instruction fetches return a stream of bytes irrespective of address. This stream is 778 bytes long, and generates a 256-byte boot program at the start of memory (addresses 0002h-0101h). The last two bytes of the instruction stream (D3 F8) write 0 ('end of boot sequence') to port 0F8h. The next instruction fetch will then be from address 0002h, the start of the boot program.

There are two known boot programs; one for dot-matrix PCWs, and one for daisy-wheel PCWs. They are identical except for two bytes: the code used to check if the boot sector is valid, and another byte which has presumably been altered so the checksum comes out the same.

3 RAM and paging

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.

3.1 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.

3.2 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:

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".

4 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.

4.1 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 4.2. 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.

[1] Arnold Number Two

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.

4.2 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.

5 Interrupts

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

5.1 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.

5.2 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 8.1.

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.

5.3 Serial port interrupts

The CPS8256 interface (see section 7.1) can be programmed to generate interrupts.

5.4 Daisywheel interrupts

The daisywheel printer (see section 6.2) can be programmed to interrupt when it has finished a command.

6 Printer ports

6.1 PCW8256/8512/9256/10 printer controller

6.1.1 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.

6.1.2 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.

A1,nn Move the print head a small distance. Not followed by a C0,00 command.

It is possible that this command has A0,nn / A2,nn / A3,nn variants like the A8/A9/AA/AB commands below, but A1 is the only one I have observed in the wild.

The second byte of the command gives the distance to move, in ticks divided by 12. The CP/M printer driver only uses this command for moves of less than 156 ticks.

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. It is possible that there are A5/A6/A7 variants for half-speed and reverse line feeds, but I haven't seen the PCW printer driver generate them.

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 CP/M printer driver appears to assume that between this command and the C0 00 that ends the command, the head will move 155 further ticks. For example, if it wants to move the head 216 ticks it will send a request to move by 61 ticks: A9 3D C0 00. In order to get bidirectional print to align in JOYCE, I had to split this value of 155 into 67 ticks at the start of the block and 88 ticks at the end.

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.

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. It is possible that there are AD/AE/AF variants for half-speed and reverse line feeds, but I haven't seen the PCW printer driver generate them.

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.

6.1.3 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 + 67 = 201 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 88 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.

6.1.4 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 A.

6.2 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

6.3 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 client computer in a LocoLink conversation - see section 7.4.

6.4 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.

6.5 Standalone CEN port

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

The status is read from IN (84h). Bit 0 is 1 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.

7 Communications interfaces

7.1 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.

7.1.1 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

7.1.2 CPS8256 Centronics port

The Centronics port (described in section 6.4) 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.

7.1.3 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.

7.2 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).

7.3 Early SCA interface

An older version of the SCA interface (presumably, the Mk1 as opposed to the Mk2) was not compatible at a hardware level with the CPS8256. Instead, SCA would supply a patch to CP/M; the resulting startup file would be named M14CPM3.EMS rather than J14CPM3.EMS. I don't have either the interface or the patch, so can't say how they differed.

7.4 LocoLink interface

LocoLink is used to connect the PCW to the parallel port of another computer. Usually this would be a PC, but it could also be a PcW16 and third-party software existed allowing a PCW9512 to do it (thus allowing transfers between 3" and 3.5" PCWs).

In early versions, the PC would act as a file server ("slave" in LocoLink terminology), and the PCW as the client (the "master"). A utility run on the PCW would transmit files to the PC. Later versions reversed the roles, with the PCW acting as the file server and the other computer treating it as one or more disc drives.

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+ can emulate the PCW end of a LocoLink connection.

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

7.5 PCW Linkit

The PCW Linkit interface is a similar device to the LocoLink (a 2-bit communication channel) except that both ends connect to PCW expansion ports. It appears at port 0FFh. The values are carried by the two low bits of the port; the value read by one PCW is the binary inverse of the value written by the other PCW.

The communications protocol used is outlined in Appendix C.

7.6 PCWMail

The PCWMail connection is a cable designed to be used in conjunction with the PCW's MAIL232 software to transfer files between two PCWs. Consequently, from the point of view of the PCWs it behaves like two CPS8256 interfaces connected back-to-back, with a null-modem cable in between.

7.7 Miracle Technology WS4000

According to reviews, this was a modem that connected directly to the PCW expansion port. Other models of the same modem were sold either with conventional serial interfaces, or direct interfaces for different computers such as the Spectrum.

My guess is that it emulated at least the serial port functionality of the CPS8256 and so would have appeared to the PCW as a normal serial port with a modem permanently connected to it.

7.8 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.

8 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.

8.1 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.

8.2 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. As far as I can see from the schematic, selecting drive 3 selects both drives simultaneously.

8.3 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.

8.3.1 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".

8.3.2 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".

9 Hard drives

JOYCE does not emulate real PCW hard drives, so this information is not verified.

9.1 Cirtech Gem

Physically, the Gem drive (at least the one I have) is a Seagate ST351A/X IDE drive, jumpered for XT mode. If the jumpers are set for AT mode it is possible to connect it to a modern PC and access the data at a sector level. Jumper settings for the ST351A/X can be found on Dell's website: <http://support.euro.dell.com/support/edocs/dta/18814/00000003.htm>.

In XTA mode the drive has four registers, mimicking an XT hard drive controller. On the PCW they can be found at ports 0A8h to 0ABh. The information below is adapted from the Interrupt List entry for ports 0320h-0323h, which is where the registers appear on a PC. For full programming information, see the IBM PC technical reference, pages 1-187 to 1-201.

0A8h Data register.

0A9h When read: controller status. When written: Reset controller.

0AAh When read: controller DIP switches. When written: Generate controller-select pulse.

0ABh When written: DMA and interrupt mask (not used on the PCW)

In addition, the Gem interface has a 4k boot ROM. It would appear that on initial startup, memory accesses with A7 reset go to the boot ROM, while memory accesses with A7 set go to the PCW mainboard. So the 4k ROM is mapped into memory from 0000-007F, 0100-017F, 0200-027F, ..., 1F00-1F7F.

The first thing the GEM boot rom does is to mimic the normal PCW boot process by copying the standard boot image (section 2) into memory. It does this by repeated memory reads (from address 80h, so with A7 set) until the sequence D3 F8 [OUT (0F8h),A] is encountered. Then it executes its own write to that port to switch to normal execution, and copies itself into RAM.

While the ROM is paged in, all I/O port accesses use 16-bit I/O [OUT (C),A style], with the top 8 bits of the address set to 80h. The first instruction after the boot ROM has been copied into RAM is IN A,(0A9h), with A=0. Presumably this I/O read, with the top 8 bits of the address all 0, pages the boot ROM out.

The format of the Gem drive boot track is documented in Appendix D.

In a Gem-2 (mirrored) configuration, the 'main' drive is drive 1 (ie, jumpered as slave) and the 'backup' drive is drive 0 (ie, jumpered as master).

9.1.1 Cirtech Gem (ATA)

I've also seen (but not really studied) a Gem drive based on a Conner Peripherals CP3024, which is ATA rather than XTA. The ATA registers would be mapped at 0A8h-0AFh:

0A8h Data

0A9h Error / Features

0AAh Sector count

0ABh Sector number

0ACh Cylinder low

0ADh Cylinder high

0AEh Drive / Head

0AFh Status / Command

with the interface presumably converting between the 16-bit transfers expected by the drive, and the 8-bit transfers done by the Z80.

9.1.2 Cirtech Insyder / Hardpak

These are pretty much Gem drives in different form factors (the Insyder is fitted inside a PCW9512, and the Hardpak includes a 2.5" hard drive in the interface box). I have examined an Insyder, which again was an ATA drive. Before a transfer, the driver writes 0 to port 0FF94h, which may be to prepare whatever mechanism converts between 8-bit and 16-bit transfers.

On the disk, sectors are stored with a non-obvious byte arrangement. Each 512-byte sector is stored as 256 little-endian words:

- The high bytes of words 0-127 give bytes 0-127 of the sector.

- The low bytes of words 0-127 give bytes 128-255 of the sector.
- The high bytes of words 128-255 give bytes 256-383 of the sector.
- The low bytes of words 128-255 give bytes 384-511 of the sector.

9.2 Cirtech Flash Drive

The Flash Drive is an adaptation of the Gem drive concept, except that both hard drive and boot ROM are replaced with one or two Intel E28F008SA flash chips. There is also a physical switch to disable booting from the Flash Drive, allowing the PCW to be booted as normal if the drive does not contain a valid boot image. In normal use the drive is read-only; special utilities are supplied to erase it and copy files to it.

The drive is paged in by a read from I/O port 0xxD8h, with the high byte of the address specifying which 32k bank to select. This mechanism would allow a Flash Drive of up to 8Mb. However the supplied utilities are only aware of the 1Mb / 2Mb configurations. Reading from I/O port 0xxD9h pages the drive out again.

When the drive is paged in, the selected memory bank is visible at 00h-7Fh, 0100h-017Fh, 0200h-027Fh ... 0FF00h-0FF7Fh. Code that is accessing the drive, and any transfer buffers, must therefore be between addresses 0xx80h-0xxFFh.

Because of the physical design of the interface, the data lines for the first chip are connected in reverse order (the Z80's bit 0 is the chip's bit 7). This doesn't matter for data reads and writes, but the supplied utilities have to cope with this when issuing commands and reading the status register. For example, to check the 'low voltage' indicator in the status register the utilities have to look at bit 5 for the first chip, bit 3 for the second.

If the 'autoboot' switch is turned on, then when the PCW is booted bank 0 of the drive is mapped into memory, and therefore should contain a valid boot program. As with the Gem drive, its first action is to read the standard boot image into memory, by repeated memory reads from 80h. Similarly, port I/O is performed in 16-bit style with the bit 15 of the port address set until the standard boot image has been read, and 8-bit style subsequently. On the Gem drive this would page out the boot ROM, but as the Flash drive does not have a separate boot ROM I suspect this may just be legacy code.

The format of the Flash drive boot track is documented in Appendix E.

9.3 ASD PCWHD10 / PCWHD20

I haven't seen one of these drives, so this information comes from disassembling the driver (ASD.FID). The drive uses an ATA interface, and appears to make 8-bit transfers. The registers are:

0A0h DataA

0A1h Error / Features

0A2h Sector count

0A3h Sector number

0A4h Cylinder low

0A5h Cylinder high

0A6h Drive / Head

0A7h Status / Command

The review in '8000 Plus' says that the ASD drive can have 1-6 partitions, but the driver I have doesn't support this.

9.4 Timatic Winchester Expansion Box

This is a 20Mb Seagate ST225, connected to the PCW through a SASI interface. The controller appears to be something similar to an NCR 5380, mapped into PCW I/O space at ports 0A0h-0A7h:

0A0h SCSI data read/write

0A1h Initiator command register

0A2h Mode register

0A3h Target command register

0A4h SCSI bus status register / select register

0A5h Bus and status register

0A6h Start DMA target receive / Input data register (neither used by PCW)

0A7h Start DMA initiator receive / reset parity (neither used by PCW)

The format of the partition table is described in Appendix F.

9.5 Other hard drives

'8000 Plus' has also reviewed several other hard drives, for which I have no programming information:

ACC Hard Disc: No information available.

Vortex System 2000: A Miniscribe 8450 XT. I've seen this described as connected through a SCSI interface, but the 8450 XT drive uses XTA, not SCSI (though apparently some versions used the same 50-pin connector as SCSI drives). The same drive was sold for other systems, notably the Amiga, Atari ST and IBM PC, using an appropriate "personality module" to interface to the host computer.

Cirtech Diamond: An earlier version of the Gem, but based on a SCSI interface rather than IDE. Among its features was the ability to connect up to seven PCWs to a single drive, taking advantage of the ability of the SCSI bus to support multiple controllers. I have no more technical details, but Cirtech's SCSI drives for Apple computers used an NCR 5380 so they may have used the same controller for the Diamond.

10 Keyboard

The keyboard appears as a memory-mapped device at 3FF0h-3FFFh in memory block 3. The first 12 bytes give the status of pressed keys:

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
3FF0h	Keypad 2	Keypad 3	Keypad 6	Keypad 9	Paste	F1/F2	Keypad 0	F3/F4
3FF1h	Keypad 1	Keypad 5	Keypad 4	Keypad 8	Copy	Cut	PTR	Exit
3FF2h	[+]	½	Shift (both)	Keypad 7	>	Return	]	Del->
3FF3h	.	?	;	<	P	[	-	=
3FF4h	,	M	K	L	I	O	9	0
3FF5h	Space	N	J	H	Y	U	7	8
3FF6h	V	B	F	G	T	R	S	6
3FF7h	X	C	D	S	W	E	3	4
3FF8h	Z	Shift Lock	A	Tab	Q	Stop	2	1
3FF9h	<-Del		J1: Fire 1	J1: Fire 2	J1: Right	J1: Left	J1:Down	J1:Up
3FFAh	Alt	Keypad .	Keypad Enter	F7/F8	[-]	Cancel	Extra	F5/F6
3FFBh			J2: Fire 1	J2: Fire 2	J2: Right	J2: Left	J2: Down	J2: Up

Using the key numbering scheme in the PCW manual:

- 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 entries marked J1 and J2 are for keyboard joysticks (see below).

The last four bytes contain controller status in bits 6 and 7. Bits 0-5 of each byte are used (by analogy with the two joystick entries) to provide keyboard combinations that may be useful as joysticks.

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
3FFCh (with LK2 present)			(KP Enter) (Shift)	(Space) (S)	(KP 0) (D)	(Exit) (A)	(F1/F2) (X)	(F3/F4) (W)
3FFDh	~LK2	Shift Lock LED	(Space)	(KP 2)	(KP 3)	(KP 1)	(KP .)	(KP 5)
3FFEh	LK3	LK1	(Shift)	(Space)	W R P] ; > . ½	Q E O [L < , /	Z X C V B N M	A S D F G H J
3FFFh	Update flag	Ticker	(Shift)	(Space)	W R P [S F X V	Q E O [A D Z C	B N M , . /½	H J K L ; < >

3FFCh gives an inverted T pattern centred on F1. If link LK2 is present, it will also respond to a W/A/D/X diamond, with S as Fire 1 and Shift as Fire 2.

3FFDh maps to the numeric keypad.

3FFDh bit 6 reports the state of the Shift Lock LED; 1 if lit, else 0.

3FFDh bit 7 is 0 if LK2 is present, 1 if not.

3FFEh maps ASDFGHJ to up, ZXCVBNM to down, QEO[L</, to left, WRP];>.½ to right, Space to Fire 1, Shift to Fire 2.

3FFEh bit 6 is 1 if LK1 is present, 0 if not. However, if LK1 is present the keyboard enters a self-test mode and transmits test patterns rather than these flags.

3FFEh bit 7 is 1 if LK3 is present, 0 if not.

3FFFh maps HJKL;<> to up, BNM,./½ to down, QEO[ADZC to left, WRP[SFXV to right, Space to Fire 1, Shift to Fire 2.

3FFFh bit 6 toggles with each update from the keyboard to the PCW.

3FFFh bit 7 is 1 if the keyboard is currently transmitting its state to the PCW, 0 if it is scanning its keys.

If no keyboard is present, all 16 bytes of the memory map are zero.

10.1 Keyboard matrix

The physical layout of the keyboard matrix is given in the service manual:

Pin	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
P10	Keypad 2	Keypad 3	Keypad 6	Keypad 9	Paste	F1/F2	Keypad 0	F3/F4
P11	Keypad 1	Keypad 5	Keypad 4	Keypad 8	Copy	Cut	PTR	Exit
P12	[+]	½	Shift (both)	Keypad 7	>	Return	]	Del->
P13	.	?	;	<	P	[	-	=
P14	,	M	K	L	I	O	9	0
P15	Space	N	J	H	Y	U	7	8
P16	V	B	F	G	T	R	S	6
P17	X	C	D	S	W	E	3	4
P24	Z	Shift Lock	A	Tab	Q	Stop	2	1
P25	<-Del	Keypad .	Keypad Enter	F7/F8	[-]	Cancel	Extra	F5/F6
P26	Alt		J1: Fire 1	J1: Fire 2	J1: Right	J1: Left	J1: Down	J1: Up
P27			J2: Fire 1	J2: Fire 2	J2: Right	J2: Left	J2: Down	J2: Up

10.2 Keyboard Links

The keyboard has three option links. By default they are all disconnected.

LK1: If connected, puts the keyboard into a test mode in which it repeatedly sends various patterns of data to the PCW. The Shift Lock LED will be constantly lit (or, more accurately, blinking faster than you can see).

LK2: If connected, pressing Shift does not cancel Shift Lock. Also enables W/A/D/X joystick, and resets bit 7 of byte 3FFDh.

LK3: If connected, sets bit 7 of byte 3FFEh. Has no other effects.

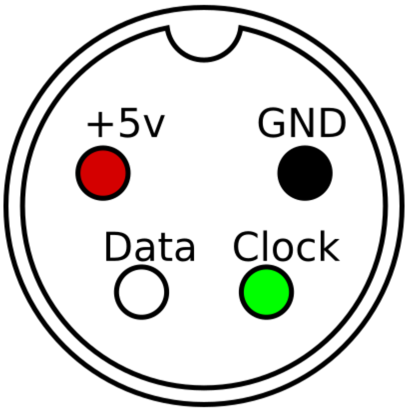
10.3 Keyboard Joystick(s)

The key numbering scheme on the PCW exactly matches the memory map until the gap at key 72. It also exactly matches the keyboard matrix schematic in the PCW9512 service manual - until key 72.

As shown above, the keyboard schematic 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.

My guess is that in the original ANT design, the keyboard matrix and the table in memory matched. When the PCW keyboard was created, the membrane was re-designed to move the now-redundant joysticks to the end of the layout, and the keyboard controller microcode rewritten to swap bytes 9 and 10 (bits 0-6) so their positions in the memory map didn't change.

10.4 Physical connection



The pinout above shows the keyboard socket on the PCW, seen from the outside of the case. The voltages used for signalling appear to be less than TTL normal, though the PCW9512 (and probably the other models) can take signals at TTL levels without apparent harm.

10.4.1 Wire protocol

By default, the data and clock lines are high. To send a bit, the keyboard drives the data line low or high, pulls the clock line low, and then a little later returns them both to high. Exact timings are unknown, though the PCW motherboard seems happy to accept timings similar to those used by the PC1512 keyboard (set data, wait for 5µs, set clock, wait for 5µs, return both lines to high, wait for 40µs).

A keycode is 12 bits long, with the most significant bit first. The first four bits are the offset in the memory map (0-0Fh), and the last eight bits are the value to be placed into memory at that address. The PCW keyboard toggles the DATA line twice before sending each word, but the gate array at the PCW end doesn't seem to need this.

When transmitting its state, the controller sends a 17-keycode packet. The first word is byte 0Fh, with the top bit set to 1; then bytes 0-0Eh; then byte 0Fh again, with the top bit set to 0.

11 Pointing Devices

11.1 AMX Mouse

This mouse appears at ports 0A0h-0A3h. The interface is built around an 8255 PPI:

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

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

0A2h (port C)
Input: 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.
Output: Stop Press writes 0FFh followed by 0, presumably to reset the counters.

0A3h (PPI mode). Stop Press writes 93h at startup (Basic I/O mode, ports A and B input, port C low 4 bits input, high 4 bits output).

If at startup the value read from port 0A2h is 10h, Stop Press switches to accessing the AMX mouse through ports 80h-83h. Possibly this is to detect an ASD hard drive and avoid a conflict.

11.2 Kempston Mouse

This mouse appears at ports 0D0h-0D4h:

0D0h gives X position, 0-255. The same value can also be read from 0D2h.

0D1h gives Y position, 0-255. The same value can also be read from 0D3h.

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

11.3 Keymouse

The Keymouse connects between the PCW and the keyboard. It uses the same mechanism as the keyboard to appear as a memory-mapped device, replacing four of the keyboard joystick bytes:

3FFBh Bits 6-0: horizontal movement counter. Bit 7: Middle mouse button pressed.

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

3FFDh Bits 4-0: low bits of vertical movement counter.

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

11.4 Electric Studio light pen

The lightpen appears at ports 0A6h and 0A7h. Its position is calculated using a 12-bit position counter and two flag bits:

<table>
<tr><th>Port</th><th>Bit 7</th><th>Bit 6</th><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th>Bit 0</th></tr>
<tr><td>0A6h</td><td colspan="8">Position: Low 8 bits</td></tr>
<tr><td>0A7h</td><td>0</td><td>V. Retrace</td><td>H. Retrace</td><td>0</td><td colspan="4">Position: High 4 bits</td></tr>
</table>

The GSX driver (DDESP.PRL) checks for the presence of the light pen by reading port 0A7h and checking the result is not 0FFh. NEWSDESK does a slightly different check, enabling the screen (OUT 0xF8, 0x07) and immediately checking that bits 6 and 7 of port 0A7h are zero.

The 12-bit position counter remains unchanged until the pen sensor is activated. When it is activated, the counter starts decrementing, and continues to decrement until the next horizontal retrace. Thus the vertical position of the pen is detected by manually counting the number of horizontal retraces before it is triggered; the horizontal position by reading the position counter.

An outline of the code in DDESP that reads the light pen position:

1. Wait for port 0FDh bit 6 to go high (ie, if the printer is doing anything, wait for it to finish).
2. Disable interrupts.
3. Wait for vertical retrace to go high (2 successive reads of port 0A7h have bit 6 set)
4. Save the initial 12-bit value of the position register.
5. Wait for vertical retrace to go low (2 successive reads of port 0A7h have bit 6 clear)
6. Wait for horizontal retrace (read of port 0A7h with bit 5 set, followed by read with bit 5 clear).
7. If vertical retrace is now high, finish.
8. If current position value differs from previous value, the sensor has been triggered. Record X = (previous position - current position), Y = current Y coordinate.
9. Increment Y.
10. Repeat steps 6-9 for 288 screen lines, or until vertical retrace goes high.
11. Re-enable interrupts.

This may produce more than one pair of coordinates; if so, the light pen position is taken to be their average.

12 Joysticks

12.1 Keyboard joystick(s)

These are described in section 10.3.

12.2 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.

12.3 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.

12.4 Cascade (JoyceStick) 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.

12.5 DKTronics interface

The DKTronics interface is a register on the DKTronics sound board's AY-3-8912 sound generator chip. To read it, write 0Eh to port 0AAh and then read port A9h. This is a bidirectional port, and can also be used for output.

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.

The interface was supplied with a joystick driver DEFJOY.COM, which converted joystick movements into keypresses. This works by patching the running copy of CP/M to add code to the interrupt handler; every 16 interrupts, it checks the joystick position and injects the appropriate scancodes into the keyboard queue. Since DEFJOY.COM uses hardcoded addresses and memory areas within CP/M, it only works on PCW8000 CP/M with BIOS 1.4.

13 Sound Generators

13.1 DKTronics sound generator

According to posts on www.amstrad.es, if I read Google Translate aright, this interface is based on an AY-3-8912, accessed using the following ports:

A9 Read currently selected register

AA Select register

AB Write currently selected register

As mentioned above, the joystick port is register 0Eh.

14 Networking

14.1 Multilink

The MULTILINK system was a ring-topology peer-to-peer network designed by Nine Tiles Computer Systems. It supported a range of 8-bit and 16-bit computers, including the PCW as well as the Apricot PC and BBC Micro. It could be used to access networked hard drives, designated FILESTORE or AMSTORE.

To its host computer, a Multilink interface is designed to appear roughly like an 8251 USART. It uses two I/O ports:

0A6h Control / status. On input:

Bit	Meaning
7	Always 1 (8251 DSR)
6-2	Always 0 (8251 error flags)
1	Set if byte waiting to send to PCW
0	Set if ready to receive byte from PCW

On output, write 48h (ie, set the 8251 Break and Reset signals) to reset the card. Other bits should be 0.

0A7h Data (read/write)

On power-up, if there is no network connection, the card will send the byte sequence 00 90 99 00 to the PCW. This signifies 'ring broken at current station'.

A full description of the character sequences used to control the card, and the meanings of the bytes it returns, are outside the scope of this document.

14.2 ASD Uninet

This was announced by ASD in May 1987 as “an improved version of the Northern Computers Network system” which could allow a PCW to act as a server. “Northern Computers Network system” probably means Multilink, since the PCW Multilink card was made by Northern Computers.

14.3 Amstrel

HM Systems produced a version of their MINSTREL server for Amstrad networking. This was an S-100 system running TurboDOS, which combined the features of MP/M and CP/Net. PCWs could thus either use it as a CP/Net fileserver, or behave as dumb terminals to 8-bit or 16-bit programs running on the server. It is likely that PCW workstations would have been connected to the server by RS232, rather than with specialised networking hardware.

15 Scanners / Image Capture

JOYCE does not emulate scanners or other image capture devices, so this section is sketchy at best.

15.1 MasterScan

The MasterScan is an optical sensor which clips to the print head of the PCW dot-matrix printer. The user is instructed to remove the printer ribbon, and the MasterScan software then programs the printer to draw a vertical line in graphics mode with minimal line spacing. As the head goes to and fro, the sensor is polled to scan the image.

The sensor is read from port 0DFh - the bottom bit is 0 for light, 1 for dark.

The user guide for ’The Desktop Publisher’ by Database Software mentions a “Dart Scanner” which probably refers to the MasterScan, since it was sold by Dart Electronics.

15.2 ProScan

ProScan is a package comprising a Naksha hand scanner, an interface to connect it to the PCW, and software to drive it. The software can also use an Amstrad FX9000T / FX9000AT fax machine as the image capture device.

The interface is accessed on ports 0E9h-0EFh. Ports 0EAh-0EDh appear to select what image capture device is to be used:

0EAh Fax (FX1)

0EBh Fax (FX2)

0ECh Hand scanner

0EDh Unknown, but there is at least some support for it in the software; possibly a second type of hand scanner.

15.3 Digitisers

Video capture devices were made by Electric Studio and Rombo. I don't have any software or drivers for these, so can't give any detail on how they operated.

15.4 OCR input

Issue 6 of "8000 Plus" describes the Oberon Omni-Reader, which scans printed text and supplies it to a computer. This uses RS232 rather than a dedicated interface, so on a PCW input from the Omni-Reader would come through the CPS8256 serial port.

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?

[2] <http://www.seasip.info/Cpm/software/amstrad.html>

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.

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 1.00-1.20

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. As in CP/M, offsets are based on the font address (1181h).

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 (MATRIX.STD must be prepared to be
     loaded anywhere in memory).
DE = address of a byte to which character width should be written.
```

On return, the registers should be:

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

The character bitmap generation code should then be called with:

```
 A = character
BC = address of this routine
DE = address of 24-byte buffer in which to store the generated character 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 written 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 v1.40-1.50

These versions of LocoScript doesn't have a MATRIX.STD file. Instead they have a PRINTER.JOY. The format of the data is the same, but 1D6Fh needs to be added to all offsets.

A.8 Matrix fonts in LocoScript 2-4

In LocoScript 2, the dot-matrix printer font is stored in MATRIX.PRI (in versions prior to 2.20) and in MATRIX.#*xx* font files (in versions 2.12 and later).

There are three versions of these files:

1.1 LocoScript 2.00-2.11. Support for 480 characters. User-defined characters not supported.

2.1 LocoScript 2.12-2.19. Support for 480 characters. User-defined characters supported.

3.1 LocoScript 2.20+. Support for 512 characters. User-defined characters supported.

The format of the files is broadly similar; they start with a 128-byte header:

Offset	Value	Notes
00h	3 ASCII bytes	File Type: 'CHR' for a font, 'PRI' for the printer driver itself
03h	Byte	Major version, 1-3
04h	Byte	Minor version, always 1
05h	90 ASCII bytes	Identity: 3 lines each of 30 ASCII bytes
5Fh	Word	Link number
61h	Word	16-bit checksum of file excluding header
63h	Word	Length of file in 128-byte records, excluding header
65h	12 ASCII bytes	Style name
71h	Byte	Style pitch
72h	Byte	Font type / flags
73h	8 ASCII bytes	Driver
7Bh	Word	Offset to the font within this file
7Dh	Word	Options
7Fh	Byte	Header checksum

The font type / flags byte is decoded as follows:

Bits 0-2: Font type. Possible values include:

1 PCW8000 dot-matrix

2 PCW9000 daisywheel

3 External printer (dot-matrix, daisywheel, inkjet etc.)

4 External 24-pin dot-matrix printer

Bit 3: Set for a font, clear for a printer driver.

Bit 4: Set for files containing a dot-matrix font supported by LocoChar (major version 2 or 3).

For a font file, the offset to the font is 0x80 (ie, the font immediately follows the header). For a printer driver file, the driver comes first and the font further on.

Taking 0 as the start of the font, it has this format:

Offset	Value	Notes
0000h	12 ASCII bytes	Character set name
000Ch	2 bytes	Unknown
000Eh	Byte	Bit 7 set if all bitmaps are uncompressed?
000Fh	Byte	Bit 6 set if screen characters have been redefined
0010h	240 bytes	PS widths table for 480 characters.
0100h	128 bytes	(v2+) 16 8x8 bitmaps for screen characters
0180h	1922 bytes (v1/v2) 2050 bytes (v3)	Index table: four bytes per character, plus one extra word at the end
	124 bytes	NLQ character bitmaps (second pass)
	124 bytes	NLQ character bitmaps (first pass)
	124 bytes	Draft character bitmaps
	Varies	Character pattern data

Note that even in a v3 file (which defines 512 characters) there are only 480 entries in the PS widths table. Characters 256-287 are skipped.

The 4-byte index entry for each character is formed:

DW offset The top 3 bits of the offset give the meaning of the remaining 13:

Offset >= 0E000h This is an accented character. Bits 13-0 of the offset give the base character number. Bits 4-0 of the flags byte give the accent character number. For example, character 200 (â) is stored as 61 E0 12 6B. Offset 0E061h means the base character is 61h ('a'). Flags 12h mean the accent number is 2 (circumflex, character 3).

Offset 0C000h-0DFFFh This is a duplicate character. Bits 13-0 of the offset give the duplicate character. For example, character 352 (capital Alpha) is stored as 41 C0 FF 57. Offset 0C041h means the shape from character 41h ('A') is used.

Offset 08000h-0BFFFh This is an uncompressed character (normally this would apply only to user-defined characters). Bits 13-0 of the offset give the address of its bitmaps relative to the start of the index table. The bitmaps are a standard size: 11 bytes for draft, 48 for NLQ.

Offset 00000h-03FFFh This is a standard character. Bits 13-0 of the offset give the address of its bitmaps relative to the start of the index table. The low 4 bits of byte 3 ('extra') give length of the 'draft' bitmap, and the remainder is the 'NLQ' bitmap.

DB flags

Bit 7: Character shifted down. Shifted-down characters are printed on the bottom 8 printer pins, rather than the top 8; intended for characters with descenders.

Bit 6: Shift character down when superimposing an accent.

Bit 5: Character design cannot be copied in LocoChar. Set on the large mathematical symbols (large Pi, large Sigma etc.) because they are drawn in two halves, top and bottom.

Bits 4-0: For an accented character (offset >= 0E000h), the accent number. Otherwise:

Bit 4: When superimposing an accent, it replaces a dot (eg, on 'i' or 'j'). Any non-blank line from the accent will completely replace the corresponding line from the character.

Bits 3-0: Proportional spacing columns minus 11

DB extra For a character where offset >= 0C000h, the low 8 bits of the end of the previous character's bitmap. Otherwise:

Bits 7-5: Blank columns at left (NLQ mode)

Bit 4: Blank columns at left (draft mode)

Bits 3-0: Number of bytes in draft character shape

This code will compute the number of bytes in a character shape:

```
if (char[n].offset & 0x4000)
{
return 0; /* This character is a reference */
}
else if (char[n+1].offset & 0x4000)
{
/* Next character is a reference */
    unsigned offset2 = (char[n].offset & 0x1F00) | char[n+1].extra;
    if (offset2 < (char[n].offset & 0x1FFF)) offset2 += 0x100;
    length = offset2 - (char[n].offset & 0x1FFF);
}
else
{
/* Next character is not a reference */
length = (char[n+1].offset & 0x1FFF) - (char[n].offset & 0x1FFF);
}
```

For compressed characters, following the offset in the index table leads you to the draft-mode pattern. 'extra' gives the length of the draft-mode pattern in bytes; add it to the offset to get the address of the NLQ pattern.

In the pattern, bit 7 of a byte is set if it should be preceded by a blank column. Bits 6-0 give a pattern number:

00h-7Bh Bitmap reference. The bitmaps begin immediately after the index table; I have called this address index_end.

NLQ (v1): There is a table of bitmaps at index_end, containing 124 words. Within each word, the high byte is for the first pass and the low byte is for the second pass.

NLQ (v2/v3): There are two sets of bitmaps, each containing 124 bytes. The bitmaps for the first pass are at index_end+7Ch and the bitmaps for the second pass are at index_end.

Draft: There is a single set of bitmaps, at index_end+0F8h.

In all the tables, within each bitmap byte bit 0 is the top pin and bit 7 is the bottom pin.

7Ch Literal (only found in NLQ patterns). Followed by two bytes giving the pin bitmaps for the second and first passes.

7Dh-7Fh Repeat. The following column is repeated value-79h times.

For uncompressed characters, following the offset in the index table leads you to 11 bytes giving the character bitmap for draft mode, followed by 48 defining it in NLQ mode.

The draft mode shape is 11 bytes. In each byte, bits 0-7 give the pin bitmap (bit 0 is the top pin, bit 7 the bottom).

The NLQ mode shape is 24 words. In each word, the high 8 bits give the pin bitmap on the first pass, and the low 8 bits the pin bitmap on the second pass.

B The LocoLink wire protocol

I have examined, or know of, the following versions of LocoLink:

1.03 / 1.04: Distributed with LocoScript PC 1.x. A file server (LLPC.EXE) is run on the PC, and a command-line client program on the PCW (LLPCW.COM) is used to send files to the PC. The connection can either be through the LocoLink hardware, or a serial port.

2.02: For use with LocoScript PC Easy and LocoScript Professional 2 Plus. The file server (LLINK202.EMS) is run on the PCW. The PCW then appears as a drive within the Disk Manager at the PC end. Serial support is dropped and the LocoLink hardware is required.

3.0 / LocoLink for Windows: This is mainly a file format conversion program, but it can also access the PCW through the LocoLink hardware. Although the hardware is the same, version 3 of the file server software is required at the PCW end. The main difference from version 2 is that version 2 shares a single drive and lets the user on the PCW choose which one is shared; version 3 shares all drives and the user on the PC chooses which to use within the LocoLink software.

LocoLink16: When importing a file using the "Import" menu option, a PcW16 can access a non-16 PCW through the LocoLink hardware. This uses very similar client code to LocoLink for Windows (to the extent that I suspect the code was ported from one platform to the other). When in use, the PcW16's "Import Document" screen gains checkboxes for "LocoLink (A/B)", allowing documents from either drive of the PCW to be read in. Again, this requires version 3 of the file server at the PCW end.

At the lowest level, the system for sending data through the LocoLink hardware seems to be the same in all known versions. The protocol appears to be symmetrical - ie, the server and the client go through the same steps to transmit a packet. However, it's easier to see what is going on at the PCW end, where the LocoLink interface presents the data directly to the CPU, than at the PC end where the parallel port interface gets slightly in the way - see section B.1.1.

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). On the wire, the sense of both lines is inverted (it's 1 if the PCW is sending 0, and vice versa). Then when the PC reads the parallel port, the BUSY signal is inverted again:

On the PCW			On the wire		On the PC		
PCW writes	Bit 1	Bit 0	BUSY	~ACK	~BUSY	~ACK	PC reads
0	0	0	1	1	0	1	040h
1	0	1	0	1	1	1	0C0h
2	1	0	1	0	0	0	00h
3	1	1	0	0	1	0	080h

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 server sends 2 and the client sends 3. In the protocol described below, this means that the server 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 16-bit 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 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 client sends a packet to the server:

Client value	Server value	Comment
3	2	Client not listening; Server listening
0	2	Client starts sending packet
0	**3**	Server acknowledges
1	3	Client ready to transmit bytes
1	**1**	Server 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 (server deduces this from byte count)
3	**3**	End of packet acknowledged
2	3	Client listening; server 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 Serial connection

In LocoLink v1.0x, it is also possible for packets to be sent and received over an RS232 serial interface. In this case the packets are sent at 9600 baud with one stop bit. The same packet format is used (type, length, data, checksum).

B.7 Startup sequence

The first packet exchange is a startup sequence.

In version 1.03 / 1.04, this is:

Sender	Type	Length	Additional bytes		
Client	07h	02h	43h	31h	
Server	55h	01h	07h		

In version 2.02, this is:

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

In LocoLink for Windows and LocoLink PcW16, the initial exchange is:

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

For versions 2.02 and later, 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.

B.8 While the link is running

The server now waits for a packet from the client, and reacts depending on what it receives. The packet types used vary depending on LocoLink version.

B.8.1 LocoLink v1.03 / v1.04

The server supports the following packet types:

14h Disconnects the link and shuts down.

21h Check space for file. Packet contains a little-endian 32-bit value (file length), followed by the filename.

2Eh Create file. Packet contains a little-endian 32-bit value (file length), followed by the ASCIIZ filename.

3Bh Append data bytes to currently open file. Packet contains the bytes to append.

48h Close file. Packet contains a word (probably file CRC).

6Fh Create directory. Packet contains ASCIIZ path name.

7Ch Find first file. Packet contains directory search flag (0 to search for directory, 1 for file) followed by ASCIIZ filename.

It returns the following packet types:

55h Result. Two bytes follow. The first is the type of the packet it is replying to, the second is an error flag (0 if succeeded, other value if failed).

62h Invalid packet type received. No bytes follow.

B.8.2 LocoLink v2.02

The server supports the following packet types:

22h Disconnects the link and shuts down.

39h DOS function call. This packet can be 1-15 bytes long. The data bytes in the packet are:

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

In theory longer versions of this packet might 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 2.x server computer provides only one disc drive.

95h Requested data (see 7Eh below)

0C3h Acknowledge data return (see 0ACh, 0DAh below)

It returns the following packet types:

50h Result. Packet length is 0-8 bytes; its payload is the values for AX, BX, CX, DX (low byte first).

67h Error. Packet length is 2 bytes; the first is the DOS error number (see INT 21h function 59h) and the second is 0.

7Eh Request data. Used where the corresponding DOS function would pass data using a pointer or DMA. The packet contains 2 bytes. The first byte is a code giving the source of the data, the second is the maximum data size. The client will reply to this with a packet of type 95h containing the requested data. The source codes are:

11h Fixed length - copied from DX in original function call

36h Fixed length - copied from DTA

5Bh Fixed length - copied from SI in original function call

80h ASCIIZ (eg: filename) - copied from DX in original function call

0A5h ASCIIZ (eg: filename) - copied from SI in original function call

0CAh ASCIIZ (eg: filename) - copied from DI in original function call

0EFh Fixed length - copied from immediately after previous parameter

0ACh 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 is the location code (as for 7Eh, but only 11h, 36h, 5Bh and 0EFh are supported), and the remainder contains the data. The client will reply to this with a 0C3h packet, the first byte of which is 0ACh.

0DAh Keep-Alive. Used when the server requires user intervention while servicing a call from the client (for example, it is displaying a "Drive not ready" error message). Expects the client to acknowledge with a 0C3h packet, the first byte of which is 0DAh. This stops the client timing out waiting for a response from the server.

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

DOS function calls 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	none	none
11h	get disc label	none	Server returns disc label if found
13h	delete disc label	none	none
16h	create disc label	none	Client sends extended FCB for labe
17h	rename disc label	none	Client sends extended FCB for rena
36h	get free space	none	none
39h	create directory	none	Client sends directory name
3Ah	remove directory	none	Client sends directory name
3Bh	set current directory	none	Client sends new directory (*3)
3Ch	create file	CX=attributes	Client sends pathname (*2)
3Dh	open file	AL=mode	Client sends pathname
3Eh	close file	BX=handle	
3Fh	read file	BX=handle, CX=count	Server sends data read
40h	write file	BX=handle, CX=count	Client sends data to write
41h	erase file	none	Client sends pathname
42h	set file pointer	AL=whence, BX=handle	
43h	get file attributes	AL=0	Client sends pathname
43h	set file attributes	AL=1, CX=attributes	Client sends pathname
44h	get device information	AL=0 BX=0 CX=0	
47h	get current directory	none	Server returns current directory (*
4Eh	find first file	AL = search attribute (*4)	Client sends search path. If successful server returns find file d
4Fh	find next file	none	none
56h	rename file		Client sends old & new pathname
57h	get timestamp	AL = 0, BX = file handle	
57h	set timestamp	AL = 1, BX = file handle	
5Bh	create file (atomic)	CX=attributes	Client sends filename

Notes:

***1** These functions always return error 5, "access denied", since under CP/M user areas cannot be created or destroyed.

***2** LocoLink v2.x expects all pathnames to be absolute, eg Z:\LETTERS\PENNY.LS2. The client must also ensure that passed file names are in upper case, as the server does not check or enforce this.

***3** The current directory set by function 3Bh is returned by function 47h but it does not affect operations in any other way since (as mentioned above) absolute paths need to be passed to all the other functions.

***4** The search attribute is passed in AL whereas in the original INT 21h API it is passed in CX.

***5** Unlike 'find first file', the result of 'find next file' is returned in the result packet rather than as a separate transfer. It consists of one word (AX) followed by bytes 0x12 onward of the find file data (but without the terminating zero byte on the filename; the caller has to deduce this using the packet length).

B.8.3 LocoLink for Windows / LocoLink 16

The client may send the following packet types:

05h Echo. Contains 0-32 bytes of additional data. The server returns a packet of type 18h containing one more byte than was sent; the first byte returned is 05h and the rest is the additional data.

6Dh Disconnects the link and shuts down.

80h DOS function call. This packet can be 1-15 bytes long. The data bytes in the packet are:

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

Note that these packets may be larger than the equivalent packets sent by LocoLink v2.02. For example, function 0Eh (log in drive) now passes a drive number, since this version of LocoLink allows the host PC to switch between the PCW's drives.

0CCh In response to a packet of type 0B9h. The packet contains the requested data.

The server may send the following packet types:

18h Acknowledgement. Sent in response to a packet of type 05h, 2Bh, 0DFh or 0F2h. The first byte is the type of the packet it is acknowledging. For type 05h there are 1-20 For type 0F2h there is a second byte of data.

2Bh Keep-Alive. The client acknowledges with a type 18h packet.

93h End of DOS function call (success). Contains two bytes: the value returned by the function in AX.

A6h End of DOS function call (failure). Contains two bytes: the DOS error code.

0B9h Server requests data from client. Contains two bytes: the first byte is a code giving the source of the data, the second is the maximum data size. The client returns the data in a type 0CCh packet. The source codes are:

11h Fixed length - copied from DX in original function call

36h Fixed length - copied from DTA

5Bh Fixed length - copied from SI in original function call

80h ASCIIZ (eg: filename) - copied from DX in original function call

0A5h ASCIIZ (eg: filename) - copied from SI in original function call

0CAh ASCIIZ (eg: filename) - copied from DI in original function call

0EFh Fixed length - copied from immediately after previous parameter

0DFh Data transfer from server to client. The first byte gives the destination of the data (using the same codes as 0B9h), the remainder of the packet is the data. Client acknowledges with a type 18h packet.

0F2h Keep-alive? Appears to include one byte of data.

Version 3 of the LocoLink server on the PCW supports the same range of DOS function calls as version 2. However LocoLink for Windows and LocoLink 16 only use functions marked +. The other functions are present but may not be thoroughly debugged (for example, trying to get or set the volume label in LocoLink 3.00 always appears to fail with a 'drive not ready' error).

Function number	Meaning	Parameters in initial packet	Additional tra
0Dh+	reset	AL = drive type DL = drive AL is 0 for CP/M, 0FFh for LocoScript	none
0Eh+	login	DL = drive	none
10h	close disc label	none	none
11h	get disc label	none	Server returns disc la
13h	delete disc label	none	none
16h	create disc label	none	Client sends extended
17h	rename disc label	none	Client sends extended F
36h	get free space	none	none
39h	create directory	none	Client sends direct
3Ah	remove directory	none	Client sends direct
3Bh	set current directory	none	Client sends new
3Ch+	create file	CX = attributes	Client sends pathn
3Dh+	open file	AL = access mode CL=0	Client sends pat
3Eh+	close file	BX = handle	none
3Fh+	read file	BX = handle, CX = count of bytes	Server sends requ
40h+	write file	BX = handle, CX = count of bytes	Client sends
41h	erase file	none	Client sends pat
42h	set file pointer	AL=whence, BX=handle	
43h	get file attributes	AL=0	Client sends pat
43h	set file attributes	AL=1, CX=attributes	Client sends pat
44h	get device information	AL=0 BX=0 CX=0	
47h	get current directory	none	Server returns current
4Eh+	find first file	CX = search attribute	Client sends sear If successful server retur
4Fh+	find next file	Possibly last find file data (*5)	none
56h	rename file		Client sends old & ne
57h	get timestamp	AL = 0, BX = file handle	
57h	set timestamp	AL = 1, BX = file handle	
5Bh	create file (atomic)	CX=attributes	Client sends fil
68h	Flush file	BX = file handle	

Notes:

***1** On LocoLink 3.00, attempts to get or change the drive label fail with a 'drive not ready' error. This is because the server requests the first 7 bytes of the extended FCB, but should request 8 (the eighth is the drive number).

***2** These functions always return error 5, "access denied", since under CP/M user areas cannot be created or destroyed.

***3** The current directory set by function 3Bh is returned by function 47h but it does not affect operations in any other way since (as mentioned above) absolute paths need to be passed to all the other functions.

***4** LocoLink v3.x expects all pathnames to be absolute, eg A:\USER.02\PENNY.LS2.

***5** The 'find next file' packet may either contain 1 byte (4Fh) or 17 bytes (4Fh, followed by bytes 0x06-0x15 from the find file data of the last file found). This is to allow depth-first search of directories; if LocoLink encounters a subdirectory, it will commence a new search to list files within that subdirectory. Once it has finished it will then send the long version of the find-next packet to resume the original search of the root directory where it left off. Bytes 0x06-0x14 of the find file data therefore need to contain enough information to identify which search is being continued.

***6** Unlike 'find first file', the result of 'find next file' is returned in the result packet rather than as a separate transfer. It consists of bytes 0x11 onward of the find file data (but without the terminating zero byte on the filename; the caller has to deduce this using the packet length).

B.9 Emulated DOS filesystems

LocoLink 2.x and LocoLink for Windows / LocoLink 16 both present the PCW's CP/M filesystem as a DOS directory tree. How this is done depends on the selected display mode, LocoScript or CP/M. In LocoLink 2.x, this is set on the PCW. In LocoLink for Windows / PcW16, LocoScript or CP/M mode is selected on the PC and passed as a parameter to the 'reset disks' function.

- In LocoScript mode, the root directory of the filesystem is empty, except for subdirectories corresponding to the eight LocoScript groups. The subdirectory names will be taken from the group names (so EXAMPLE.GRP would become a subdirectory called EXAMPLE), or GROUP.*n* if there is no .GRP file.
- In CP/M mode, user 0 becomes the root directory, and there will be up to 15 subdirectories named USER.01 to USER.15.

In LocoLink 2.x, there is also an option to hide empty user areas, which is selected by default.

LocoLink 2.x, which supports only one drive, appears to use Z:\ as the drive letter in absolute path names. LocoLink 16 uses A:\ (and presumably B:\ for the second drive, if one is present).

C The PCW Linkit Wire Protocol

The PCW Linkit interface, like LocoLink, has two lines in each direction. In this section I will call them H (high bit) and L (low bit). Unlike LocoLink, the protocol is symmetrical; either end can initiate a file transfer.

C.1 Establishing the link

As mentioned in the description of the hardware, the values read by one PCW are the inverse of those written by the other. When establishing the link, the Linkit software compensates for this by inverting the values immediately before they are written.

System 1 wants to send	Value written to port	System 2 receives
0	1	0
1	0	1

In the rest of this section, I will describe the values as seen by the software at both ends and ignore the inverted value on the wire.

The actions taken are the same at both ends of the link. They are:

- Set a count of successful cycles to 0.
- Send 0 on lines H and L.
- Enter a loop where:
  - If last value sent on line H was 0, examine incoming line L.
    - If line L is 0, take no action.
    - If line L is 1, invert the value of line H.
  - Else (last value sent on line H was 1) examine incoming line L.
    - If line L is 1, take no action.
    - If line L is 0, and the number of successful cycles is >= 10, break out of the loop.
    - If line L is 0, and the number of successful cycles is < 10, increase the number of successful cycles and invert the value of line H.
  - Read incoming line H, and write the opposite value to outgoing line L.
- Assuming the loop to have completed successfully, set a number of tries to zero. Then enter another loop where:
  - Examine incoming line H.
    - If it is the opposite of the last value written to L:
      - If the number of tries is less than 100, increase the number of tries.
      - Otherwise break out of the loop.
    - If it is the same as the last value written to L, invert the value of line L and reset the number of tries to zero.
- Finally, check that incoming line L is 0 and incoming line H is 1.

The effect of the first loop is (roughly) that each end is setting its output line L to the opposite of its incoming line H. It then toggles its output line H and verifies that the inverse pattern comes back on line L. The second then checks that at the end of the process, incoming line H is set to the opposite of outgoing line L and not changing. And the final check is that the values of the signals are being inverted in the link as expected.

C.2 Sending / receiving a file

To indicate that it is sending a file, one system inverts the value of line H. It then sends a stream of ternary data; the smallest unit of data sent is a 'trit', a number from 1 to 3.

The process for the transmitting system to send a trit is:

- Wait for incoming line H to change.
- The value being sent is 1-3. XOR that with the last value sent, and write the result to the output port.

At the receiving end, the process to receive a trit is:

- Invert the value being written to line H.
- Read incoming lines H/L, and hold this as the 'old' value.
- Wait until the value read on incoming lines H/L changes. XOR that value with the 'old' value to get the trit received, 1-3.

C.2.1 Byte encoding

Bytes sent are encoded as a sequence of trits. The encoding used is designed to favour ASCII text:

1,n,n,n Lower case letters (1,1,1,1 -> ’a’, 1,1,1,2 -> ’b’, 1,1,1,3 -> ’c’, 1,1,2,1 -> ’d’ etc.)

1,3,3,3 End of block. Followed by two bytes giving a (big-endian) checksum, and then a constant zero byte.

2,1,n,n,n Upper case letters (2,1,1,1,1 -> ’A’, 2,1,1,1,2 -> ’B’, 2,1,1,1,3 -> ’C’, 2,1,1,2,1 -> ’D’ etc.)

2,1,3,3,3 End of transmission.

2,2,n,n,n,n Characters in the 0x00-0x7F range that don’t have a shorter encoding:

```
NUL SOH STX ETX EOT ENQ ACK BEL BS VT FF SO SI
DLE DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM SUB
ESC FS GS RS US # $ / < > @ [ \ ] ^ _ ‘ { | } ~ DEL
```

2,3,n,n,n,n,n Characters in the 0x80-0xFF range.

3,1,1 Space.

3,1,2 Tab.

3,1,3 Comma.

3,2,n,n,n Linefeed, carriage return, digits and punctuation in the 0x21-0x3F range. The 27 characters so encoded are:

```
LF CR 0 1 2 3 4 5 6 7 8 9 ! ” % ’ ( ) * + - . & : ; = ?
```

3,3,n,n,n Run of identical bytes. The three trits give a value (1-27) which is the length minus 3. Followed by the byte to repeat.

C.2.2 File encoding

The file is sent as two blocks. The first is a 15-byte header:

Offset	Length	Meaning
0	2	File size / 128 (big-endian)
2	1	User number, ASCII ’0’ to ’7’ (*1)
3	8	File name
11	1	ASCII dot
12	3	File extension

***1** PCW-Linkit, being aimed at LocoScript users, only supports the first eight user areas.

Once a block is sent, the receiving system should verify the checksum and send 0 on its L line if OK, 1 if error.

If there is an error the transfer is abandoned, and the transmitting system moves to the next file (if there is one). Otherwise the file is sent as a stream of bytes, followed by another End of Block sequence. Again, once all bytes have been sent the transmitting system reads input line L to determine if there was an error.

The transmitting system will then send either the header of the next file, or End of Transmission.

D The Gem Drive System Track

The first track of the Gem drive contains a boot image, which is loaded and executed by the 4k boot ROM. The hard drive setup program provided with the drive will write this when asked to repartition the drive or update the boot loader. On the utilities diskette it is stored in a file called HD.SYS.

The boot image starts with a header, corresponding to the MBR on an IBM-formatted hard drive:

Offset	Size	Description
0	Word	Offset to boot programs table
2	Word	Offset to image of HDRIVER.FID
4	Word	Offset to the splash screen
6	Word	Total length of boot image
8	Byte	Options byte
9	17 bytes	CP/M Plus DPB (used for all partitions)
1Ah	Byte	Sectors per track
1Bh	Word	First cylinder of partition 1
1Dh	Word	First cylinder of partition 2
1Fh	Word	First cylinder of partition 3, 0 if none
21h	Word	First cylinder of partition 4, 0 if none
23h	Byte	8-bit checksum of bytes 0-22h

Note that the SPT field in the DPB at offset 9 gives sectors per *cylinder*, while the 'Sectors per track' at offset 1Ah really does give sectors per *track*. For example, a Gem drive with 5 heads and 17 sectors would have SPT = 5 * 17 = 85, while the field at offset 1Ah would hold 17.

Three bits of the options byte are used:

Bit 7 Set to load CP/M by default, clear to load LocoScript by default. Hold the ALT key when booting to load the other OS.

Bit 4 Set if this is a Gem-2 (mirrorred) system. In this configuration, sectors are read from drive 1 (the main drive) and written to both drive 1 (the main drive) and drive 0 (the backup drive). Hold the B key when booting to disable mirroring and boot from the backup drive.

Bit 0 Set if a splash screen is present

Working through the fields in order:

D.1 The boot programs table

This table contains a list of boot programs. Each entry is five bytes long:

Offset	Size	Meaning
0	Byte	8-bit checksum of the first 512 bytes of the EMS/EMT file
1	Word	Offset of boot program from start of table
3	Word	Length of boot program

The end of the table is indicated by a boot program with offset 0. If the checksum of the first 512 bytes of the EMS/EMT file does not match any of the entries in the table, the boot ROM will give a single beep and refuse to execute it.

If a matching boot program is found, it will be copied to 0D000h and run with the following parameters:

- HL = length of HDRIVER.FID. HDRIVER.FID will be in memory at 0D000h + length of boot program.
- C = boot program number (1-based).
- The selected EMS / EMT file will be in memory from 0000h up.

The task of the boot program is to inject the HDRIVER.FID image into the loaded EMS / EMT file, so that it boots from the hard drive rather than the floppy. The version of the bootloader that I have studied can do this to six variants of the CP/M EMS file, and 64 variants of the LocoScript EMS file.

D.2 The HDRIVER.FID image

This is an exact copy of the HDRIVER.FID used to access the hard drive when booting from floppy.

D.3 The splash screen

This is a 720x146 bitmap. It is stored as 19 rows, each containing 90 8x8 character cells. The first row should be blank, because it is used to draw the 55 scanlines above and below the picture.

D.4 HDRIVER.FIC image

One Insyder drive that I have examined stored a second driver after the splash screen. This was used when booting LocoScript 3.06+, with the original driver used by CP/M.

E The Flash Drive System Track

The Cirtech Flash drive begins with a boot image, which is executed by the PCW at startup if the 'Autostart' switch is on. The INSTALL program provided with the drive writes this boot image when initialising the drive. On the utilities diskette it is stored in a file called FD.SYS.

The file / boot image begins with a header:

Offset	Size	Description
0	Word	Short jump to boot code at 001Fh
2	Word	Offset to boot programs table (as for Gem drive)
4	Word	Offset to image of FDRIVER.FID
6	Word	Offset to the splash screen
8	Word	Offset to image of FDRIVER.FIC
0Ah	Byte	Default operating system. Bit 7 set for CP/M, reset for LocoScript
0Bh	17 bytes	CP/M Plus DPB for the drive
1Ch	Word	Sector number of first sector of directory
1Eh	Byte	Checksum
1Fh	3 bytes	Jump to first stage boot code at 0100h

Note that the first-stage boot code is stored in FD.SYS at 0080h, but will be mapped into memory at 0100h at boot time.

The remaining contents of the boot image (boot programs table, driver .FID file, splash screen, .FIC file) are similar to their Gem Drive counterparts.

F The Timatic Winchester Expansion Box Partition Table

The driver that I have seen Winchester Expansion Box assumes a 20Mb drive, with the partition table at logical sector 0xA28F.

At offset 1Dh in the sector is a printable list of drives, six bytes per drive. Each entry reads "#:nnM," with the last being "#:nnM." followed by CR / LF / LF / 0FFh. The driver will use this as a template for its startup message, replacing the '#' entries with the drive letters in use.

At offset 64h in the sector is a table of CP/M DPBs, 17 bytes per drive. These all report 4 128-byte sectors per track (that is, a single 512-byte sector per track) so the CP/M track number corresponds to the SASI sector number. This would only work on a drive size up to 32Mb. Following the last DPB is a 0FFh byte.

Byte 255 of the sector is set when written to the XOR of the previous 255 bytes, complemented. Thus when the first 256 bytes of the sector are XORed together, the result should be 0FFh.

Unused bytes in the sector are filled with the repeated text "C TURVEY."