

# P.R.O.S.E

## (Phil's Rudimentary Operating System Experiment)

Last updated 10-09-2011, for PROSE v03B

### Description:

PROSE is a simple command line operating system for the EZ80P by Phil Ruston. It operates in a similar manner to the freezer cartridges of the 8 bit era, offering the customary debugging and memory monitor features. It also has the some of the capabilities of a modern DOS, allowing the execution of programs direct from the command, scripts to be run, environment variables to be set up etc.

### Installation and OS file details:

PROSE loads as a normal file from the root directory of a FAT16 formatted SD card. The ROM-based bootloader looks for a file called "BOOT.EZO" and loads the file named therein. The bootloader loads this file to address 0x0A00 and jumps to location 0x0A10 - the CPU is in ADL mode at this point. If the OS file is not found, then "NO OS" flashes on screen - the user can download OS code via the serial link at this point if desired.

### Command line:

PROSE contains a set of internal debugging and disk navigation commands:

?	- Show internal command list
AVAIL	- Show available memory ranges
C	- Copy bytes in memory
CD	- Change directory
CLS	- Clear OS screen
D	- Disassemble ADL mode code
DZ	- Disassemble Z80 mode code
DEL	- Delete a file
DIR	- Show directory
ECHO	- Displays short text messages
FONT	- Change the font
F	- Fill Memory
FI	- File Info
FORMAT	- Format the (entire) SD card as a single FAT16 partition.
G	- Goto location (IE: Call a routine)
H	- Hunt in memory for hex bytes
LB	- Load Binary file to RAM from disk
M	- Show memory as hex bytes
MD	- Make a new directory
MOUNT	- Remount drives
MOUSE	- Enable the mouse driver
PEN	- Change the colours used by the OS
R	- Show CPU registers
RD	- Remove directory
RN	- Rename file or directory
RX	- Receive file from PC via serial link
SB	- Save Binary file from RAM to disk
SET	- Set an environment variable
SOUND	- Play sample data in audio RAM
T	- Show memory as ASCII text

```

TX          - Transmit bytes to PC via serial link
VERS        - Show AMOEBA and PROSE versions
VMODE       - Change the OS video mode
VOLx:       - Switch volume without altering the directory position.
:           - Put hex bytes in memory
>           - Put text in memory

```

(Full descriptions are given in the glossary at the end of this document)

### External Commands

When the command entered is not recognized internally, PROSE will look in the current volume / active directory (or follow a path, if one is specified) and then in the directories specified in the PATH environment variable (which is set to "COMMANDS" by default). If an executable file with a name matching the string entered (no extension is necessary), the file is loaded and executed (similarly, if the file is a script, it will be run).

Current list of external commands:

```

KEYMAP      - Changes the keyboard mapping for non-UK keyboards.
DATE        - Set / Show current date
FPGACFG     - Manage FPGA configuration
INPUT       - Show a prompt and get a string response from user
PTPLAY      - Protracker format music player
PCXVIEW     - Show .PCX format files (by Enzo)
PROTED      - A text editor (by Enzo)
SHOWBMP     - Displays .bmp pictures
TIME        - Set / show the time
TYPE        - Displays text files

```

### General:

- When PROSE starts it looks for FAT16 partitions and where found labels them VOL0:, VOL1: etc. (Note that PROSE cannot automatically detect card swaps - the volume list needs to be refreshed with the command 'MOUNT' if an SD card is changed.)
- Executable files have the file extension ".EZP" and script files have the extension ".PBF" - these can be omitted when launching commands or scripts at the command line.
- Throughout PROSE, numerical data is represented in hexadecimal format (apart from the free disk space reported by the DIR and FORMAT command).
- The default keymap is that of the UK, but this can be changed with the command "Keymap". Keymaps are currently supplied for the following locales: UK, Germany, Italy, USA, Portugal - others can be easily added to the keymaps folder (the format is described in the readme.txt file found there.)
- The editor window is by default 80x60 characters in text map mode (see details at end of this doc). The resolution can be changed with the command VMODE.
- To automate the application of the customizable settings, PROSE will run a batch file called "STARTUP.PBF" at start-up if present in the root directory of the SD card (this script can be aborted by pressing CTRL+C).
- Bear in mind when debugging that external commands load into memory the same as normal apps and will therefore overwrite whatever was there beforehand.

## Development:

Zilog's "ZDS II" software is recommended for development purposes. Following a project build, the .hex file output should be converted to a raw binary (with no pre-origin padding) and have the file extension .EZP appended. One way to do this is to drag the .hex file to the Windows app "hex\_to\_ezp.exe" (see the PC Apps folder of the project archive. Purebasic source code is provided to allow versions to be built for other operating systems). For a detailed walk-through concerning the use of ZDSII to make a new project in assembler or C, see the Coding Guides folder.

## Executable File Requirements:

PROSE-friendly programs should have an origin above address 0xFFFF and end before the upper limit given by the AVAIL command. Programs require the following header:

```
-----
0x00  JR skip_header      - CPU instruction: Jump past this header
0x02  db 'PRO'            - ASCII "PRO" = PROSE executable program ID
0x05  dw24 load_location  - Desired Load location (24 bit)
0x08  dw24 0              - If > 0, truncate load
0x0B  dw prose_version_req - If > 0, minimum PROSE version required
0x0D  dw amoeba_version_req - If > 0, minimum AMOEBA version required
0x0F  db ADL_mode         - Z80 (0) or ADL mode (1) program.
-----
```

This header is used when a command is entered to load the program to the correct location in RAM etc. (The relative jump at the start allows the size of the header to be increased in future if necessary.)

Programs will also need to include the list of equates and labels used throughout the EZ80P system (and Z80 Mode programs need to set the MBASE register to bits 23:16 of the load location). The easiest way to handle all this is to simply set the project's include folder location to that of the project archive's Code/Include folder (see "Settings" in ZDS II) and start the source with the following code (changing the equates to suit your program):

```
;-----
amoeba_version_req    equ 0          ; 0 = dont care
prose_version_req     equ 0          ; 0 = dont care
ADL_mode              equ 1          ; 0 = Z80 mode program, 1 if ADL mode
load_location         equ 10000h     ; 0x10000 upward

        include      'PROSE_header.asm'

;-----
```

## The Stack:

- The Small Stack Pointer is set to 0xFFFF by default when PROSE starts (and so the stack will be at the top of whatever 64KB page is being used for that program) but it can be set at will by the user program.
- The Large Stack Pointer (for ADL programs) is set the top of the top of system memory when PROSE starts.

### **Start and Return:**

When a program is executed, HL will contain the address of the first non-space character after the entered command name. Apps can scan from this point for arguments if required - when a zero is encountered, the search should be terminated.

To return to PROSE use the instruction: "JP.LIL prose\_return"

Prior to returning to PROSE, register A should be set as follows:

- A = 0: No error report is displayed.
- A = 1: A driver error 0xnn is assumed to be being returned in B (and will be reported as "DRIVER ERROR \$nn")
- A = 0xFE : A command is to be launched on return. HL = location of command string.
- A = 0xFF : PROSE should restart (though not a reset / cold start)

Other values are interpreted as standard PROSE error codes (see list at end of this document). The value in A returned by a program also sets an environment variable called "ERROR".

If a program wants to return without restarting PROSE but has changed "cosmetic" settings such as the display mode, it is possible to reset the PROSE video settings with the kernal call "kr\_os\_display" (see following section.)

PROSE logs the Z80 register set on return (as displayed with the R command). If this is not required, the program can call the routine "kr\_dont\_store\_registers".

### **Debugging:**

PROSE sets the NMI vector to a "freeze" subroutine so that programs can be stopped and the memory / registers examined (The push switch at the back/right of boxed EZ80Ps performs this function.)

### **Calling Kernal Routines:**

A Jump Table allows access to the OS routines (see list and details in the document Kernal\_Routines.txt). To call a kernal routine, set A to the routine label and do an ADL call to "prose\_kernal". EG:

```
ld a,kr_clear_screen
call.lil prose_kernal
```

A macro is set up in the include file "prose\_header.asm" allowing kernal calls to be written as, EG:

```
prose_call kr_clear_screen
```

(Remember that the CPU A register is always overwritten by kernal calls)

### Environment Variables:

PROSE environment variable ("Envvars") are global variables that persist until PROSE is restarted. The variable name and its value are both zero-terminated ASCII strings. 512 bytes are allocated for envvars and kernal calls are provided to utilize them within programs.

Envvars can also be set up on the command line with the SET command (use quotes for envvar values if there are spaces in the text). Additionally, the command "INPUT" can be used to prompt for a string and assign it to an envvar. Exiting programs place their return codes in an envvar called "ERROR" and can output other data using envvars called OUTxx (see below).

Standard PROSE message passing with Envvars:

It is sometimes useful for a program, upon exit, to pass data out for another program to use. PROSE's own commands do this via envvars called OUTxx (where xx is 00 to FF). (Whenever a PROSE command sets envvars to pass data, they start afresh at "OUT00" and persist until another program that outputs data in this way is run.)

Current PROSE commands that can output return data are: AVAIL, FI and VERS. The environment variables are generated (instead of displaying data on screen) when the argument "#" is supplied. EG: Type "AVAIL #" and then "SET" to see the environment variables created.

## Scripts:

PROSE will run a script (IE: a batch file) named "STARTUP.PBF" (if it exists in the root dir) when it starts. As mentioned, scripts can be manually launched at other times by simply entering their names as if starting a program, and can be aborted with CTRL+C.

Basic conditional branching is available using an IF / LABEL / END structure in the form:

```
IF environment_variable = ascii_string LABEL
or
IF environment_variable <> ascii_string LABEL
```

IE: If an environment variable has been initialized and its contents match (or do not match) the ascii\_string specified, then jump to LABEL.

It is possible to test just for the existence of an Envar with " = \*", EG:

```
IF environment_variable = * LABEL
```

Label locations are defined in the body of the script in square brackets at the start of lines (no other characters should follow a label).

END will stop a script at that line.

An example script:

```
ECHO "THIS IS A TEST SCRIPT.."
INPUT "DO THIS OR DO THAT? (ENTER: THIS OR THAT)" RESPONSE
IF RESPONSE = THIS SOMEPLACE
IF RESPONSE = THAT ANOTHERPLACE
ECHO "INVALID RESPONSE"
END
[SOMEPLACE]
ECHO "YOU OPTED FOR THIS!"
END
[ANOTHERPLACE]
ECHO "YOU OPTED FOR THAT!"
```

Notes:

- The environment variable name, value and label strings are currently limited to a maximum of 16 characters.
- Scripts cannot run other scripts.
- Volumes should not be changed within a script.

## Interrupts:

The eZ80P CPU has a 16bit interrupt vector table at I:0x0A - I:0x5F. As these vectors are limited to the lower 64KB of RAM, PROSE sets them to jump to a table of ADL jump instructions at 0x6f - 0xfb. The first byte of each of these locations is set by PROSE to a "C3" byte (JP instruction) and the following 3 bytes are the 24 bit address the interrupt should jump to. Therefore, the 24 bit interrupt vector table is as follows (interrupt vectors are listed in order of priority.)

\$0070	PRT0 vector	[24 bit address]	
\$0074	PRT1 vector	[24 bit address]	* used by millisecond counter routine*
\$0078	PRT2 vector	[24 bit address]	
\$007c	PRT3 vector	[24 bit address]	
\$0080	PRT4 vector	[24 bit address]	
\$0084	PRT5 vector	[24 bit address]	
\$0088	RTC vector	[24 bit address]	
\$008c	UART0 vector	[24 bit address]	
\$0090	UART1 vector	[24 bit address]	
\$0094	I2C vector	[24 bit address]	
\$0098	SPI vector	[24 bit address]	
\$009c	PORTB 0 vector	[24 bit address]	* used in PROSE for keyboard/mouse *
\$00a0	PORTB 1 vector	[24 bit address]	
\$00a4	PORTB 2 vector	[24 bit address]	
\$00a8	PORTB 3 vector	[24 bit address]	
\$00ac	PORTB 4 vector	[24 bit address]	
\$00b0	PORTB 5 vector	[24 bit address]	
\$00b4	PORTB 6 vector	[24 bit address]	
\$00b8	PORTB 7 vector	[24 bit address]	
\$00bc	PORTC 0 vector	[24 bit address]	
\$00c0	PORTC 1 vector	[24 bit address]	
\$00c4	PORTC 2 vector	[24 bit address]	
\$00c8	PORTC 3 vector	[24 bit address]	
\$00cc	PORTC 4 vector	[24 bit address]	
\$00d0	PORTC 5 vector	[24 bit address]	
\$00d4	PORTC 6 vector	[24 bit address]	
\$00d8	PORTC 7 vector	[24 bit address]	
\$00dc	PORTD 0 vector	[24 bit address]	
\$00e0	PORTD 1 vector	[24 bit address]	
\$00e4	PORTD 2 vector	[24 bit address]	
\$00e8	PORTD 3 vector	[24 bit address]	
\$00ec	PORTD 4 vector	[24 bit address]	
\$00f0	PORTD 5 vector	[24 bit address]	
\$00f4	PORTD 6 vector	[24 bit address]	
\$00f8	PORTD 7 vector	[24 bit address]	

By default PROSE writes only the 24 bit vector for PORTB 0. The PRT1 vector is written when the routine "kr\_init\_msec\_counter" is called. The other locations are uninitialized, except for their preceding JP instructions).

## NMI:

The NMI behaves a bit differently, when a NMI occurs the eZ80P jumps to address 0x66, therefore PROSE places a 0xc3 byte (JP instruction) at 0x66, followed by the 24 address it is to jump to:

\$0067 NMI vector [24 bit address] \* Freezer button / scanline interrupt \*

As the ROM area \$0-\$7ff is paged out once the OS runs, these locations can be freely accessed the user programs.

## **Internal Commands:**

(Arguments shown in square brackets are mandatory, round brackets optional.)

**AVAIL** - Show the unallocated address ranges of the three memories.

Use: AVAIL (#)

Notes: If the # argument is supplied, the following environment variables are set (instead of data being output to the display)

OUT00 - Sys RAM base  
OUT01 - Sys RAM top  
OUT02 - VRAM\_A base  
OUT03 - VRAM\_A top  
OUT04 - VRAM\_B base  
OUT05 - VRAM\_B top

**C** - Copy bytes in memory

Use: C [start\_address] [end\_address] [destination\_address]

**CD** - Change Directory / Volume

Use: CD .. (go to parent dir)  
CD / (go to root dir)  
CD subdir (change to subdir)  
CD VOLn: (change to root of volume n)  
CD VOLn:walks/silly (change volume and subdir)

**CLS** - Clear OS screen

Use: CLS

**D** - Disassemble a page of ADL mode code

Use: D (start\_address)

Notes: If no address is supplied, disassembly continues from the previous line.



**DZ** - Disassemble a page of Z80 mode code

Use: DZ (start\_address)

Notes: If no address is supplied, disassembly continues from the previous line.

The only difference between the D and DZ is that DZ disassembles with the assumption that all address references and immediate data words are 16 bit as per the original Z80 CPU (and not 24 as per the EZ80).

**DEL**- Delete File

Use: DEL filename (name must be a file, not a dir)

EG: DEL VOL0:tests/somefile.txt

**DIR** - Show Directory Listing

Use: DIR (path)

EG: DIR tests/mydir

**ECHO** - Show a line of text

Use: ECHO "STRING"

Notes: Quotes must be used

ECHO "" moves the cursor down one line.

**F** - Fill Memory with a specific value

Use: F [start\_address] [end\_address] [fill\_byte]

**FI** - Show file information.

Use: FI [filename] (#)

EG: FI commands/time.ezp

Notes: If the file is not a program, only the file length is shown.

If the # argument is supplied, the following environment variables are set (instead of data being displayed)

OUT00 - File length  
OUT01 - File Location  
OUT02 - File Truncate Value  
OUT03 - Minimum PROSE version required  
OUT04 - Minimum AMOEBA version required  
OUT05 - ADL mode "1" or Z80 mode "0"

If the # argument is supplied, FI does not return any error codes (EG: file not found). This can be used with the "= \*" test of the IF command to silently check for the existence of files in a script.

EG:

FI somefile.txt # (sets OUT00=filelength, if it exists)  
IF OUT00 = \* label

**FONT** - Changes the OS font

Use: FONT [font filename]

Notes: Looks in the specified directory and then in the root/fonts directory for the font file.

EG: FONT spectrum.fnt  
FONT myfonts/c64.fnt

#### Font file description:

Fonts are 1024 byte 1-bit bitmap files. The 256 8x8 pixel characters are in a simple sequential format: Bytes 0-7 define character 00, bytes 8-15 define character 01 etc. A utility is provided to convert .bmp format files.

**FORMAT** - Formats a volume or entire disk to FAT16

Use: `FORMAT [volume or device_name] (label)`

Notes: "volume" = vol0: to vol7:

"Device\_name" is that used by the driver (EG: "SD\_CARD").

A card needs to be partitioned if it is to contain more than one volume. This can be done with the PROSE util 'DISKPART.EZP' or on a Linux PC.

Four partitions are allowed - the partitioning system used by PROSE simply uses the 4 entries in the Master Boot Record, not any Microsoft-style extended DOS partition/logical DOS drive scheme. As such Windows will only recognize the first partition on a card.

FAT16 formats 2GB maximum, larger cards or partitions are simply truncated.

**G** - Goto location (IE: Call a routine)

Use: `G [address] (args)`

Notes: When G is used to run code, HL = the address of the first non-space character after the address.

"G 0" resets the system

**H** - Hunt in memory for bytes

Use: `[start_address] [end_address] [byte1] (byte2) (byte3) etc`

**LB** - Loads bytes to RAM from current volume.

Use: `LB [filename] [address]`

Notes: A path can be supplied in the filename

EG: `LB my_apps/test/somefile.bin 10000`

**M** - Show memory as hex bytes

Use: `M (address)`

Note: You can edit the displayed data and press return to update it in memory as the string is displayed with the ':' prefix

**MD** - Make new directory

Use: MD [new subdir]

EG: MD vol0:apps/new

**MOUNT** - Rescans for connected drives

Use: MOUNT

**PEN** - Changes the PROSE colour scheme

Use: PEN [pen\_colour] (paper\_colour) (up to 16 palette\_entries)

Notes: Pen\_Colour = nm where 'n' is the 8x8 font character's background colour and 'm' is the foreground colour.

Paper\_colour = n, where n is the colour (from the list below) that appears where no characters exist following a clear screen etc.

Palette entries = a list up to 16 24bit words that redefine the RGB values of the palette entries.

By default, the following colours are set:

- 0 - black
- 1 - blue
- 2 - red
- 3 - magenta
- 4 - green
- 5 - cyan
- 6 - yellow
- 7 - white
- 8 - dark grey
- 9 - mid grey
- a - light grey
- b - orange
- c - light blue
- d - light green
- e - brown
- f - pink

**R** - Show registers - Updated on exit from external programs or NMI button.

Use: R

**RD** - Remove directory

Use: RD [subdir]

Notes: argument must be a directory, not a file

EG: RD commands/test (removes "test" subdir of "commands")

**RN** - Rename file or directory

Use: RN [original filename] [new filename - no path]

EG: RN vol0:myfiles/this.txt that.txt

**RX** - Receive data from serial comms port (place it in RAM) and optionally execute it as code.

Use: RX [filename] [address]

Notes: If filename is "\*" whatever file is sent is accepted.

If filename is "!", the file is downloaded and immediately run.  
HL is set to the location of any argument data following the "!"  
The file must be an executable program, but note that truncation  
info in the header is ignored.

If the filename is ">", the file is not loaded to memory, but  
immediately copied to the current volume / folder.

**SB** - Save bytes from RAM to current volume

Use: SB [filename] [address] [length]

EG: SB my\_apps/somefile.bin 10000 800

**SET** - Sets an environment variable

Use: Set [name] = [string]

Notes: If no arguments are given, the currently set up environment variables are listed.

Reserved names are:

ERROR, which holds the value of the error return of the last file ("00" if no error)

OUTxx (where xx is 00-FF) - used to pass data to a following program.

PATH - a list of root subdirectories, seperated by a space where PROSE looks for executables and scripts.

**SOUND** - [location] [length] (frequency) (volume) (channels) (loop)  
Plays bytes memory (VRAM B) as audio clip

Notes:

"Location" refers to an address between c00000 and c7ffff (VRAM B) IE: the memory that is designated audio RAM in AMOEBA (shared with Sprites).

(NB: Only the lowest 19 bits are read by the hardware so a value of 0-7ffff can be supplied and this will automatically be assumed to be an offset from the start of VRAM\_B.)

"Frequency" is a constant derived by the formula:

```
(Desired Freq)
(-----) X 65556) - 1
( 48828    )
```

"Channels" selects the audio channels on which the sound will play. Each channel is assigned one bit (bit 0 = channel 0, bit 1 = channel 1 etc..) Unselected channels are not affected and will continue doing whatever they were doing beforehand.

"loop" is 0 or 1. If 0, the sample is played one. If 1 the sample loops around the start and plays continually.

If frequency, volume, channels or loop are omitted the following default values are used.

```
frequency: FFFFh, IE: 48828 Hz
volume    : 40h,   IE: Full volume
channels  : 11h,   IE: Channels 0 and 4, (one left and one right)
loop      : 01h,   IE: Sound plays continually
```

If no parameters are supplied, all the channels are silenced.

**T** - Show memory as ASCII text

Use: T (address)

Note: you can edit the displayed data and press return to update it in memory as the string is displayed with the > prefix)

If no address is supplied, the output continues from the current line.

**TX** - Transmit bytes from RAM to serial comms port (using Serial Link app)

Use: TX [filename] [address] [length]

Notes: Choose "Receive File" on the Serial Link utility before pressing Enter on this command.

**VERS** - Shows the OS and Hardware version numbers

Use: VERS (#)

Notes: If the # argument is supplied, the following environment variables are set:

OUT00 - PROSE version  
OUT01 - AMOEBA version

**VMODE** - change the resolution of the OS display window

Use: VMODE [n]

Note: If n = 0, display = 80x60  
If n = 1, display = 80x30  
If n = 2, display = 40x60  
If n = 3, display = 40x30

**VOLx:** - Swap to a different volume without changing the currently active directory on that volume.

Use: Volx:

Notes: x = 0 to 9

The command CD can also change the volume but will go to the root of the volume.

**>** - Put text in memory

Use: T [address] ["text"]

**:** - Put hex bytes in memory

Use : [address] [byte1] (byte2) (byte3) etc..

**?** - List commands

Use: ?



## **External Commands:**

**DATE** - Set / Show the date

USE: Date (dd:mm:yyyy)

**INPUT** - Request a string from the user and set it as an environment variable.

USE: Input ["PROMPT"] [label]

Notes: Environment labels and strings are limited to 16 characters.

PROMPT needs to be in quotes

**KEYMAP** - Change the keymap

Use: KEYMAP [keymap file]

Keymap looks in the current directory and then in the root/keymaps directory for the keymap file. The current keymap list is:

UK.bin - UK (default)

US.bin - USA

DE.bin - Germany

IT.bin - Italy

PT.bin - Portugal

### **Keymap File Format:**

Keymap files are "PS/2 set 2" scancode-to-ASCII translation tables. They can contain two or three "banks" of the translation table: one for unshifted keys, one for shifted keys and optionally one for Alt-modified keys. Each bank is 98 bytes long, covering scancodes \$00-\$61

**TIME** - Set / Show the time

USE: TIME (hh:mm:sec)

## **Apps and Utils:**

**FPGACFG** - Manage the FPGA configuration

Use: FPGACFG [B/C/L/R/W] [arg1] [arg2]

Notes: B = Set the power on boot slot to [arg1]  
C = Configure from slot [arg1]  
L = List the contents of all EEPROM slots  
W = Write filename [arg1] to slot [arg2]

If the no filename is supplied, the config file is expected via serial link.

**PARTCARD** - Partitions SD cards for use with PROSE.

Use: PARTCARD

**PCXVIEW** - Display a PCX format graphics file

Use: PCXVIEW [filename]

**PTPLAY** - Play a Protracker format tune

User: PTPLAY [filename]

**PROTED** - Edit a text file

Use: PROTED (filename)

Note: See the Docs/Apps folder for full info.

**SHOWBMP** - Display a 256 colour BMP format graphics file

Use: SHOWBMP [filename]

**TYPE** - Display a text file

Use: TYPE [filename]

## **PROSE Error Codes:**

\$01 - Driver error. The actual error code from the device driver is returned in register B.

\$80 - Aborted  
\$81 - No data  
\$82 - Bad data  
\$83 - Time out  
\$84 - Address bad  
\$85 - Comms error  
\$86 - Checksum error  
\$87 - Incorrect file  
\$88 - Out of range  
\$89 - Unsupported Device  
\$8a - Device not detected  
\$8b - Device error  
\$8c - Script error  
\$8d - Missing args  
\$8e - Cannot allocate memory

\$c1 - Disk full  
\$c2 - File not found  
\$c3 - (Root) dir table is full  
\$c4 - Directory requested is actually a file  
\$c5 - Cant delete dir, it is not empty  
\$c6 - Not a file  
\$c7 - File length is zero  
\$c8 - Out of memory  
\$c9 - Filename already exists  
\$ca - Already at root directory  
\$cb - Directory not found  
\$cc - Requested bytes beyond EOF  
\$cd - Invalid filename  
\$ce - Unknown/incorrect disk format  
\$cf - Invalid volume  
\$d0 - Device not present  
\$d1 - Directory not found  
\$d2 - End of directory list  
\$d3 - Device does not use MBR  
\$d4 - Cant find volume label  
\$d5 - Sector out of range

## **EZ80P Memory resources used by PROSE:**

- Sys RAM : 0x0-0x00FFFF, 0x07FE00-0x07FFFF
- VRAM\_A : 0x800000-0x806580 (font, charmap)
- VRAM\_B : 0xC7FE00-0xC7FFFF (pointer sprite)

## **EZ80 Resources used by PROSE:**

- UART0
- RTC / TIMER0
- TIMER1 - used for millisecond counter (when enabled)
- PortB 0 IRQ (keyboard and mouse from AMOEBA)
- NMI (freezer / scanline IRQ from AMOEBA)

## PROSE Video Mode:

The OS window uses AMOEBA's character mapped mode. Each pair of bytes fetched from VRAM refers to a character and its colour attribute. The first byte selects a character tile, the second byte selects the pixel colour (3:0) and background colour (7:4) for that 8x8 tile.

- The font is located at VRAM\_A: 0x800000
- The character map is located at: 0x804000

See the hardware manual for more information about character modes.

## Serial Data Protocol:

(As used by RX/TX commands and PC-side Serial Link App)

### To send a file:

1. Create and send a file header packet (see format below)
2. Wait for 2 ASCII bytes from receiver - if "OK" goto step 3, if "WW" wait longer, anything else = error.
3. Send file byte packet of 256 bytes.
4. Send 2 byte CRC checksum of packet
5. Wait for 2 ASCII bytes from receiver - if "OK" goto step 3, if "WW" wait longer, anything else = error.
6. Goto step 3 until all bytes sent (file must be padded to 256 byte packet size)

### To Receive a file:

1. Wait for file header.
2. Test CRC checksum of header when it arrives.)
3. Check filename is that of file required if necessary.)
4. Send ASCII bytes "OK" if checksum/filename are OK, "WW" if sender needs to wait, else "XX" for error.
5. Receive 256 byte file packet
6. Receive 2 byte CRC of packet
7. Test CRC of packet
8. Goto step 4 until all bytes received.

### Header Format - first 256 byte packet

0x00 - 0x0F: ASCII filename  
0x10 - 0x13: Length of file (little endian longword)  
0x14 - 0x1F: ASCII "Z80P.FHEADER" (12 chars)  
0x20 - 0xFF: All must be zero

### File format:

Bytes from the file, sent in 256 byte packets followed by a 2 byte checksum word.

### CRC computation:

The calculation is described as a "standard CRC-CCITT that uses polynomial \$1021" and produces a 16 bit output, the Z80 code (slow, unoptimized) to generate it is as follows:

```

;--Z80 code to make CRC -----
; makes checksum in HL, src addr = DE, length = C bytes

crc_checksum

crcloop      ld hl,$ffff
             ld a,(de)
             xor h
             ld h,a
             ld b,8
             add hl,hl
             jr nc,crcnext
             ld a,h
             xor 10h
             ld h,a
             ld a,l
             xor 21h
             ld l,a
             djnz crcbyte
             inc de
             dec c
             jr nz,crcloop
             ret

;-----
```

### Storage Device and Driver Tables:

PROSE can use additional storage device drivers either at source level or by loading to memory. A driver must allow data to be read and written in 512 byte sectors with a 32-bit address and contain the following routines:

- Initialize device, return ID and total capacity in sectors
- Read sector into buffer from device
- Write sector from buffer to device

All routines should clear the Zero flag on return if the operation was successful. If the zero flag is not zero, a driver-specific error code is returned in A. The initialization routine should return with BC:DE (16 bit mode register) set to the total capacity of the device in sectors, and HL set to the location of a zero-terminated ASCII string to identify the device (this string is used for reference only).

Device driver code must start with the following structure:

```
$00 - JP initialize / get ID routine
$04 - JP read sector routine
$08 - JP write sector routine
$0c - ASCII name of device type (null terminated)
```

If a device driver is included in the PROSE source code, its location should be added to the "driver\_table" list and it should use the standard PROSE variables:

"sector\_buffer" - 512 bytes

"sector\_lba0" - 1 byte LBA of desired sector LSB

"sector\_lba1" 1 byte

"sector\_lba2" 1 byte

"sector\_lba3" - 1 byte LBA of desired sector MSB

If an externally loaded user RAM based driver is to be used, the above locations can be obtained with the kernel routine "kr\_get\_disk\_sector\_ptr" (HL = location of LSB of 32-bit sector address, DE = location of sector buffer). The location of the start of a driver can be put in the driver table by first calling "kr\_get\_device\_info" to obtain the location of the Driver table (see below) scanning for the first non-zero 24bit location (as each device driver entry takes 3 bytes), and placing the address of the driver at that location. A maximum of 4 drivers can be used.

#### Driver table:

\$00 - Driver 0 address (The EZ80P's SD card driver)  
\$03 - Driver 1 address  
\$06 - Driver 2 address  
\$09 - Driver 3 address

#### Host Device Hardware Info:

The address of the hardware information table is returned in HL by the kernel routine "kr\_get\_device\_info" Each device entry is 32 bytes long and contains the following data:

#### OFFSET | DATA:

0x00 - Device's assigned driver number  
0x01 - Device's TOTAL capacity in sectors (4 bytes)  
0x05 - Zero terminated hardware name (22 ASCII bytes max followed by \$00)  
0x1c - Remaining bytes to \$1F currently unused

#### Volume Mount List

The address of the volume mount list is returned in HL by the kernel routine "kr\_volume\_info". Each volume entry is 16 bytes long which contains:

#### OFFSET | DATA

0x00 - 1 = Volume present, else 0 (This doesn't mean it's a valid FAT16 volume!)  
0x01 - Volume's host driver number  
0x02 - [reserved]  
0x03 - [reserved]  
0x04 - Volume's total capacity in sectors (3 bytes)  
0x07 - Partition number on host drive (0/1/2/3)  
0x08 - Offset in sectors from MBR to partition boot sector (2 words, little endian)  
0x0c - [reserved]  
0x0d - [reserved]  
0x0e - [reserved]  
0x0f - [reserved]