Documentation for the ZX Spectrum emulator

'Z80' version 2.01

(3/5/93)

by

Gerton Lunter
P.O. Box 2535
NL-9704 CM Groningen
The Netherlands

LaTeX conversion by Lars Köller

Contents

1 INTRODUCTION, REGISTRATION, GENERAL INFORMATION

1.1 Some general remarks

This is the documentation for 'Z80', a Sinclair ZX Spectrum 48/128 emulator. This program turns your PC into a Spectrum. To make you read on...

> The program emulates a Spectrum 48K model 2 or 3 or Spectrum 128K, is highly compatible with the real machines, includes the Interface I, supports Microdrives, tape files, the RS232 channel, '128 sound through internal speaker or Adlib compatible soundcard, able to save and load every Spectrum program directly to and from tape, even able to load speed-saved programs, supporting digital and analogue PC joysticks and four common Spectrum joysticks, Z80 processor emulation including the R register, inofficial opcodes and flags, accurate timing of individual instructions, control over the emulated Spectrum's speed, and all that while requiring only a PC-XT with 512K with CGA, Hercules, EGA or VGA; offering conversion programs to convert between various emulators' Snapshot formats and to read from Disciple and Plus D diskettes, to create .PCX and .GIF files of Spectrum screen dumps, an English manual,...

There is much to tell and explain in this documentation. First of all the emulator itself must run, and uses your PC's resources. It is not really a demanding program, but there are some things that need attention. These technicalities are dealt with in section 2.1.

Some general things about the emulator are explained in section 2.2. If you read 2.1 and 2.2, you'll be able to do most of the things you probably ever want to do. But to exploit all of its possibilities (and oh, it can do so much!), you will really have to read it all.

The Spectrum has a number of ways to communicate with the outside world, like the obvious keyboard and the screen, but also the microdrives, the tape interface, the beeper, the sound chip of the Spectrum 128, the Kempston joystick, and the RS232 channel of the Interface I and Spectrum 128. All these channels can be used to communicate with PC channels in some way; for instance the keyboard is connected to the PC keyboard (sounds obvious) and the tape I/O can be routed to a file, as well as to a physical tape recorder. All these things are explained in the rest of chapter 2. Paragraph 8 of that chapter contains a number of suggestions how to transfer Spectrum programs to the PC.

For our own Spectrums Johan Muizelaar and I built a piece of hardware we called the SamRam (which has nothing to do with the SAM coupe, by the way!). It contains a monitor program and software to make snapshots of programs. It's still very useful and I still use it a lot, although the physical SamRam doesn't work anymore. An explanation of its functions is to be found in chapter 3.

Some things peculiar to the Spectrum, not specific to this program but useful to know are collected in chapter 4. It contains for instance a table of Spectrum keywords and the key combination to get it; unfortunately this information is not printed on standard PC keyboards!

There are some interesting, rather unknown technical facts about the Spectrum that I discovered while debugging the emulator. As much as I could think of is contained in the final chapter. You don't need to read this chapter to use the emulator; if you don't find it interesting then skip it, but I think programmers will like it.

A remark about copyrights. The source files are not public domain, and you may not use them in other PC-based Spectrum emulators. Also, the information in this documentation file, especially the info in the final chapter (except for the file-format info in the final section) may not be used for that purpose. But you're free to use the info for Spectrum emulators for other machines, provided that whenever you do so you should name the source.

For Spectrum software, utilities, other emulators for PC's as well as other computers, and other Spectrum related software, you can call the Spectrum Emulator support BBS in Groningen:

Tatort BBS Groningen
050-264840
(+31-50-264840)
v22, v22bis, v32, v32bis, MNP2-5, v42, v42bis (300-14400 baud)

At the time of writing the BBS is open 24 hours a day, but this is subject to change. Please try calling between 22:00 and 9:00 local time first.

If you have access to Internet, you can find several Spectrum emulators in a directory of for instance wuarchive.wustl.edu (take a look in .../systems/sinclair and .../msdos/emulators) or nic.funet.fi. And if you want to get in touch with me, my email address is gerton@rcondw.rug.nl.

1.2 Registering - sounds interesting!

First of all, this program is shareware. That means that you're encouraged to give it away to others, but if you do be sure not to alter the files. Also please don't add things to the archive file. Shareware means that you may try the program some time; if you like it you should register for it. The shareware version of the emulator consists of the following files:

Z80.EXE	– The emulator
Z80.INI	– Default initialisation file
Z80.DOC	– The documentation file
Z80.TEX	– LaTeX documentation file, by Lars Köller
Z80.PS	– PostScript documentation file, from the .TEX one
Z80.PIF	– Program Info File to run 'Z80' under Windows 3.1
Z80.ICO	– Windows icon
ROMS.BIN	– Various ROM images
LAYOUT.SCR	– Keyboard lay-out help screen
GETRS.COM	– Utility to receive blocks through RS232 lead
SAVESPEC.Z80	– Utility to send .Z80 files from Spectrum to PC
DIAGRAM.Z80	– Circuit diagram for tape interface, and calibration
Z80FAQ.DOC	– Frequently asked questions - and answers!
NEW.DOC	– The What's New file

The shareware version of the emulator program is not fully functional. It cannot be slowed down, and it can't load programs from tape. All other functions work the same in both versions. If you register, you will receive the fully functional emulator together with the following utilities:

CONVERT – a general conversion program: can list out BASIC and tranlate it back, produce .GIF or .PCX files from screen-dumps, translate Spectrum ASCII (CR) to PC ASCII (CR/LF), and some other things.

CONVZ80 – Translates various snapshot and tape formats of other Spectrum emulators into each other. Can handle Arnt Gulbrandsen's (JPP) .SNA format, Pedro Gimeno's (VGASPEC and SPECTRUM) .SP format and Kevin J. Phairs' (SPECEM) .PRG format. It can also handle tape files of SPECEM and L. Rindt and E. Brukner's emulator ZX.

DISCIPLE – Reads DISCiPLE and Plus D diskettes, both 3.5" and 5.25". It translates the 48K and 128K snapshot files to .Z80 snapshots, and ordinary files and screen snapshots to .TAP tape files.

Z802TAP	– Converts a .Z80 snapshot, 48K or 128K, to a .TAP file which can be loaded into the emulator and saved to tape by the next utility:
TAP2TAPE	– Saves the contents of a .TAP file back to tape, to load it into an ordinary Spectrum.
Z80DUMP	– Shows the contents of the header of a .Z80 file.

You will also receive the source files of the emulator, the above utilities and the SamRam, and you'll be kept informed about future updates.

The registration fee is 20 US$, or 15 British pounds, or 35 German Marks, or 35 Dutch guilders, or some of your local (hard) currency of about that amount. Now there are several way to get the money to me. In order of preference:

1. Simply send banknotes.
2. From Europe, send a Eurocheque of HFL 35,–
3. Send a postal money-order (Works fine from e.g. Italy and Spain)
4. Send a bank cheque. Please add the equivalent of 20 Dutch guilders, for that's the amount the banks charge for drawing foreign cheques.

If you're sending a Eurocheque, make sure you fill it in completely (don't forget the number at the back!) and fill in 'Groningen' for the place. If you don't send Dutch currency, or don't fill it in completely, or fill in a foreign city, the banks charge me fifteen to twenty guilders to cash the cheque.

For Dutch users, the fee is HFL 25,–. In Nederland gaat het betalen het gemakkelijkst via de giro. Maak het bedrag over op giro 59.45.263 t.n.v. G.A. Lunter, Groningen. Zorg er wel voor dat uw naam en adres vermeld staan! (vooral als u Girotel gebruikt)

Send the money, together with your name and address to:

Gerton Lunter
P.O. Box 2535
NL-9704 CM Groningen
The Netherlands

You'll get the files on a 3.5" DD disk by default, but you can also get in on 5.25 inch disks if you want.

Registrations can also be handled by B G Services in the UK if this is more convenient. The cost is the same (15 British pounds). Payment can be by cheque or postal order made payable to B G Services. The address is:

B G Services
64 Roebuck Road
Chessington
Surrey KT9 1JX

Telephone enquiries on 081 397 0763, Fax 081 391 0744.

For registrations in the Czech Republic, I recommend to contact JIMAZ, who will provide details on registering in the local currency:

JIMAZ s.r.o.
Hermanova 37
170 00 Praha 7
phone: +42 2 379 498
fax: +42 2 378 103

1.3 Other emulators

There are several other Spectrum emulators, both for the PC and other computers. The list below is partly due to Carlo Delhez (the QL emulators) and partly copied from Arnt Gulbrandsen's documentation of his JPP. I don't think the list is complete, so if you know more Spectrum emulators, for any computer, please let me know.
For the PC:

- JPP, by Arnt Gulbrandsen (Norway). Faster than mine (but according to an OUTLET review slower on some boards), by using a very smart decoding technique, but requires a 80386 or '486 processor. Is less compatible than Z80. Uses the .SNA snapshot format. Needs VGA. Not many extra features.

- VGASPEC, by Alberto Olloqui (Spain). Needs VGA and 80286. Quite slow, and crashes on quite a lot of programs. Uses the .SP snapshot format. Allows ROM pokes. This program is an illegal pre-release of SPECTRUM, by Pedro Gimeno.

- SPECTRUM, by Pedro Gimeno (Spain). Uses another .SP snapshot format. Has a tape interface. Also quite slow. Allows changing the rom.

- SP, by J. Swiatek and K. Makowski (Poland). Cannot load or save snapshots, but can load programs using `LOAD ""` via a file called TAPE_ZX.SPC. Crashes many programs; even basic behaves weird sometimes. Has a built-in monitor, but no documentation. No border.

- SPECEM, by Kevin J. Phair (Ireland). Also allows rom changes. Displays the registers on screen. Can save and load directly from disk using LOAD/SAVE "filename" in BASIC. Loads .PRG snapshots, but cannot save them. Emulates a Multiface I.

- ZX, by L. Rindt and E. Brukner (Czech Republic). Haven't tested its compatibility thoroughly, but one of the games supplied didn't respond well to the keyboard, while it did work on Z80 after conversion. Good tape file support including headerless files, almost identical to the multiple .TAP file mode of Z80. Somewhat slower than Z80. Includes program to load from tape and convert to tape file. No documentation at all.

For the Sinclair QL:

- SPECTATOR by Carlo Delhez, The Netherlands; shareware; supports tape-files, Microdrives, RS232, Z80 snapshots, MBF snapshots and Disciple (SNP) snapshots; utilities to convert Disciple, Beta and Opus disks enclosed.

- ZM-1/2/3/4 by Ergon Development, Italy; ZM-1 is shareware, ZM-2/3/4 are commercial. They all support tape-files and Z80 snapshots, some support Microdrives and RS232; contain a utility to transfer programs from tape via a Spectrum to the QL.

- ZX by Andew Lavrov, CIS; shareware; supports tape-files, MBF snapshots en Z80 snapshots; utility to read from Spectrum tapes (and write them).

Spectator, ZM-1 and ZX are all about as fast (approximately 30 to 40on a 16 MHz MC68000 machine). ZM-2/3 are faster, but this at the cost of compatibility. ZM-4 is not an emulator, but a real-time Z80-compiler: very fast and seems to be compatible as well.

For the Amiga:

- Spectrum, by Peter McGavin. Very good, JPP is based to a large extent on it. Needs about a 25MHz machine to run at full speed. Has tape support.

- KGB. I haven't seen it. A bit slower than Peter's, and the version Peter saw wouldn't work on the Amiga 3000.

- An Italian emulator which I don't know the name of. Excellent compatibility, rather fast. May be shareware.

- Several unreleased emulators. Peter knows more about them.

For the Atari ST/TT:

- One, called Spectrum. Don't know anything about it, but the doc file is written in quite the worst English I've seen. [This is Arnt speaking — I've seen worse! GAL] Available by anonymous ftp from terminator.cc.umich.edu.

- There's another one in the make, to be released very soon as one of the programmers told me, written by Markus Oberhumer and other(s).

For the Acorn Archimedes:

- A company called Arxe wrote one, intended to be commercial but never released because Amstrad wouldn't permit Arxe to enclose the ROM.
- Someone called D. Lawrence wrote another, or maybe the same. This one is floating around but nobody has any documentation. I don't know what its status is. Runs at about 70% of Spectrum speed on an ARM2, not quite perfect graphics emulation.

For the Commodore 64:

- The Whitby Software Spectrum simulator is a rewrite of the Spectrum Basic. It will not run machine-code programs. I don't know whether it's PD, shareware, or commercial. Quite good. (Responds nicely to a POKE 23659,0)

All emulators for PC, and some for the Atari, Amiga and QL are available on the support BBS.

There are also emulators available for the ZX81. Carlo Delhez, who also wrote a Spectrum emulator for the QL, wrote the ZX81 emulators XTricator (for the QL) and XTender (for PC's). These programs can also be downloaded from the support BBS.

1.4 Thanks

From the very first beginning in november 1988, when I wrote the first lines of code, Johan Muizelaar has been a very demanding and critical user, being only satisfied when it was perfect. And quite a few things I would never have started working on if he hadn't insisted that I would...

Secondly, I have to thank Brian Gaff, who is now handling the UK registrations for nearly a year, and besides doing lots of PR for my program there helped me with many things, especially with the DISCiPLE conversion program.

Finally, I'd like to thank

- Lars Köller for transforming the plain ASCII doc to LaTeX,
- Thomas Franke for translating the entire Dutch v1.45 manual into German,
- Carlo Delhez for information on the '128 and several other things,
- Andre Mostert for some more '128 info and info on EMS memory,

- Walter Prins for many '128 programs and a nice African chat,
- Marco Holmer for making the program such a big hit at the HCC dagen,
- Henk de Groot, for finding and helping me work around a bug in A86 version 3.22,
- Arnt Gulbrandsen for pointing out to me that it is not necessary to clear a register when it contains 0,
- Ruud Zandbergen for his digital joystick interface,
- Ettore de Simone for finding a noisy bug, and for some very wise remarks about theological issues,
- Jan Garnier for providing the chips to reanimate my real Spectrum,
- Rudy Biesma and Tonnie Stap for providing info on the DISCiPLE disk formats,
- Hugh McLenaghan for his very valuable help on the DISCiPLE program,
- Burkhard Taige for various bug reports on it,
- Ian Cull for enhancing the DISCiPLE program and fixing two early bugs,
- Bert Lenarts for information on the AZERTY keyboard, and
- Andre Brus for writing the most enthousiastic letter I've ever read!

2 THE EMULATOR

2.1 Starting the emulator

The emulator will work on any PC with at least 512K memory, with a VGA, EGA, Hercules, CGA or Plantronics video adapter. If available, it will also use EMS memory, an Adlib compatible soundcard, and an analogue or digital joystick.

The emulator will first read in the switches that are given in the Z80.INI file. You can enter switches there in the same way you would on the command line. Lines starting with a % sign will be ignored.

After any switches, you may specify a snapshot file on the command line. This file will then be loaded and executed directly. The extension .Z80 is not necessary. The emulator will also read .SNA files (the snapshot format of, amongst others, Arnt Gulbrandsen's JPP); you don't have to convert them to .Z80 files (but you may want to to save disk space).

The emulator tries to figure out what hardware is available, and uses things as it finds it. Most of the time this will work without you having to tell it anything, but if you have to you can override the defaults by putting switches on the command line. Switches that you use often can be put in the Z80.INI file. If you give a switch a second time, for instance if it is also in the Z80.INI file, it will disable it again.

If you are using Hercules, try starting the emulator with -xh on the command line. The emulator will use a non-standard Hercules mode to display a full-screen Spectrum picture. You may need to calibrate your monitor to make the image steady.

If you're using Plantronics, try -p and -q to see which gives the best result.

Some black-and-white VGA monitors only display one of the three RGB colours (green most of the times), resulting in several Spectrum colours becoming indistinguishable. Use -xb to use grey tones instead of colours.

If you're using a Trident VGA with version 3 BIOS, you may see the picture compressed at the top of the screen, while the bottom half contains vertical white lines. This is due to a bug in the Trident VGA Bios. Start the emulator with the switch -xv to get a full picture.

If you haven't got EMS memory, the page swapping of the Spectrum 128 cannot be emulated exactly. Most programs will work - although quite slowly because page swapping will take much time without EMS - but some may crash. On 386 and 486 machines you can emulate EMS by software, using EMM386 for instance. Of all the EMS emulators I've tried (that's three or four) QEMM was by far the fastest, but the EMM386 supplied with the new DOS 6 seems to be about as fast. A slow EMS emulator can degrade the performance of the '128 emulation significantly! Some computers have hardware EMS capabilities, some '286 boards for instance. Refer to your own documentation for details.

And don't use hard disk based EMS emulators: the Spectrum emulator will drive your hard disk nuts!

There are a few Spectrum programs that have an odd stack pointer, and run over the ram/rom boundary, for instance Deep Strike. This crashed version 1.45 of the emulator. The bug has been removed in version 2: if the emulator tries to read a word at FFFF, the processor generates an INT 0D interrupt and the emulator will handle it correctly. However, this won't work when an EMS emulator is installed that puts the 386 or 486 processor in virtual 8086 mode. You can test all this by typing

```
CLEAR 65535:POKE 65535,0: RETURN
```

in Basic, and the emulator will lock up if it runs in virtual mode. There is no simple solution to this problem, but luckily it doesn't happen often. If it does, the easiest way to to solve it is to change the Spectrum program so that it uses an even SP — this is always possible, but not always easy!

A very few programs (the only examples known to me are Fireman and Thing) are quite sensitive to the relative actual execution speed of emulated Z80 instructions, and crash if it isn't exactly right. If you slow down the emulator, these program will run fine, because then individual instructions are more carefully timed.

The Spectrum 128 has a built-in sound chip. If you have an Adlib compatible soundcard installed, the Spectrum 128 sound will be played through the Adlib card. If you haven't, the loudest of the three sound channels will be played through the internal PC speaker. Sometimes the effect is quite nice, sometimes it is horrible, but it's all I can do on a standard PC. If you don't want to have the Spectrum 128 sound played through the internal speaker, use the switch -xi. If you don't want the Adlib card to be used (for instance to hear the sound through the internal speaker) use -xa.

If you're using the Pro-Audio Spectrum 16 sound card, do not install the resident FM.EXE program; it causes problems with the emulator. Do make sure that MVSOUND.SYS is installed in your CONFIG.SYS file, to make the Pro-Audio Spectrum 16 Adlib compatible.

The noise channels of the Spectrum 128 sound chip can work on different frequencies, whereas the FM chips of the Adlib card cannot. However, if your Soundblaster is equipped with CMS chips, the noise frequency can be programmed. Specify -xc to use the CMS chips. (These chips are not available on Soundblaster Pro cards, and neither on most Soundblaster clones).

If you're living in Belgium or France, you are probably using an AZERTY keyboard. Specifying -xz on the command line will make all letter keys and many punctuation keys work in the right way.

If the emulator erroneously detects an analogue or digital joystick, use the switch -kk.

It may be annoying to have to press Num-Lock every time you use the Spectrum 128 (because otherwise you'll have to use Shift with the cursor keys to move the menu bar). To make the emulator press shift by default every time you use the PC cursor keys in '128 mode, use the switch -xs. If you press Num-Lock now (in '128 mode), the shift-key won't be pressed. The 48K modes are not affected by this switch.

The emulator can now also be run under Windows 3.1! However, you cannot use the tape interface, Real mode doesn't work anymore, and the keyboard is not emulated as well as usual, because I have to scan it using BIOS calls. Be sure not to set the keyboard repeat rate too low; an initial delay of 250 ms followed by 10 keys a second will do, but don't make it slower. Some key combinations may not work, such as ALT 4 for $. So if you want to use the emulator seriously then you shouldn't run it under Windows, but it's nice to see three Spectrums run simultaneously...If you let the emulator run full-screen you may use EGA or VGA, if you want to run it windowed you'll probably have to use CGA, because the virtual video display driver of Windows cannot handle the VGA mode I use (although it's only a standard text mode!). You'll probably want other default settings of some parameters (such as the video mode) if you run the emulator under Windows; the emulator will always use the .INI file in the directory of the Z80.EXE file so the other switches must be put on the command line, in a .PIF file. An example .PIF file (which runs the emulator in windowed CGA mode) is supplied.

Since the execution speed of a program running under Windows heavily depends on other processes, the emulator doesn't try to measure how fast it is running under Windows. It says it runs at 100%, and you can slow it down in the usual way, but the percentage is now relative to the maximum speed, and has nothing to do with the actual execution speed.

The emulator will automatically detect whether Windows is running, and act appropriately. To run the emulator in Windows compatibility mode in a normal DOS environment, use -xw.

When running the emulator under Desqview, use -e for EGA mode display.

To run the emulator with a different rom than the standard one, you can specify a rom image file on the command line. Use the switch -xr file, where 'file' is the name of the image file. This file should be exactly 16384 bytes long. It will of course not be used in Spectrum 128 or SamRam mode.

The emulator 'ZX' by Rindt and Bruckner comes with several roms, stored in their tape format. You can convert these files to .TAP files, and then load them in the normal way, but to run the emulator with these roms you need the bare 16K binary image file. To extract it from the rom files, type the following at the DOS prompt:

`C:\debug rom.000`	(or other rom file (of 16406 bytes))
`-m 115 L 4000,100`	(move the rom down, overwrite header)
`-rcx`	(new length of exactly 16K bytes)
`CX 4016`	
`4000`	
`-n rom000.bin`	(or some other name)
`-w`	(write it)
`Writing 04000 bytes`	
`-q`	(and quit)

Then start the emulator with `-xr rom000.bin` on the command line to use the rom. It will only affect the normal 48K modes; the SamRam and 128K modes will always use their own roms.

These are the most important switches that you have to specify when you start the emulator. Most of the other switches are used to select default values for various things which can also be changed when the emulator is started. Some useful things to select are default directories for .Z80, .TAP and .MDR files; these will be explained below.

2.2 Using the emulator

When the emulator starts, you'll see the usual Spectrum copyright message appear on screen. Pressing F1 will pop up a small help screen that explains the function of the function keys and various other special keys.

By pressing F10, you enter the main menu of the emulator. Most of the menu options can be chosen directly by pressing another function key. The only exception is X, Extra functions, for which no function keys were available anymore. If you're somewhere deep in the menu structure of the main menu, pressing ESC will get you one level higher most of the time. Pressing F10 will get you back to the main menu.

The 'Select Hardware' menu option sits under function key F9. There are five configurations you can choose: a normal Spectrum 48K with or without Interface I, a Spectrum 128K with or without Interface I, and a Spectrum with Interface I and SamRam. Switching to another mode will by default reset the Spectrum. If you don't want this to happen, press CTRL-ENTER instead of ENTER when you've made your choice. It cannot be guaranteed however that the Spectrum won't crash or behave weirdly, for obvious reasons. Going from a Spectrum 128 to a normal Spectrum will almost always crash it, except if you enter the SPECTRUM command before switching.

To use SamRam's monitor on a 128 program, switch the hardware from the main menu, and generate an NMI (Extra functions - N) before returning to the emulator. This will often work, but you can't return to the program without crashing it.

On a real Spectrum 128, the menu bar of the startup screen is moved using the cursor keys on the '128 keyboard. These keys simultaneously press a normal cursor key (5,6,7 or 8) and shift. So you can shift the menu bar with shift-6 and shift-7. As is already said above, it is possible to use the PC cursor keys for this; you have to select Cursor joystick emulation (which is selected by default) and press Num-Lock once to have the PC-cursor keys press the Spectrum Shift key too. You could also specify -xs on the command line (or put it in the Z80.INI file) to make the PC cursor keys by default press shift for you in '128 mode.

The Save and Load Program options (F2 and F3) will save the whole state of the Spectrum and some of the emulators' settings to a .Z80 snapshot file. It will pack the data somewhat, so that the length of the file varies. The amount of memory saved depends on the current hardware mode; 48K for normal Spectrum, 80K for SamRam, and 128K for Spectrum 128. The settings that are saved are those that are program dependent, for instance which joystick emulation is used, and more technical settings like those of the R register, LDIR and Issue 2 emulation, double interrupt frequency and video synchronisation. These are explained below.

Loading a .Z80 file will cause several settings to be changed. Resetting the Spectrum will not reset these settings to their default values! Especially the joystick emulation setting change can be confusing, so keep track of that.

The Change Settings menu pops up if you press F4. You can do many things here, and I won't explain them all here. The I and O options can be used to redirect the RS232 output; see paragraph 2.6 for information on this.

R: R - register emulation, and

L: LDIR emulation

are seldom needed. For remarks on these options see chapter 5, and paragraph 2.8.

2: Issue 2 emulation will turn the emulated Spectrum in an Issue 2 Spectrum. (This option also works, but is ridiculous, in Spectrum 128 mode). Some very old programs (Spinads) will not respond to the keyboard properly on Issue 3 Spectrums, and for these programs this option was added. Seldom needed.

F: fast flash makes flashing go twice as fast. Not very useful.

S: sound enables you to turn off all sound, useful for late-night playing.

D: double interrupt frequency is useful for slow machines, as some programs will run faster when this option is on. If you're typing in a BASIC program on a slow machine, always turn this on, since the keyboard, which is polled by an interrupt routine, will respond much better. On the other hand, some programs will crash with this option active.

V: video synchronisation is used to remove the flickering of moving characters in some programs. You can choose between Normal, High and Low. Normal works well for almost all programs; Ghosts and Goblins and Zynaps look much better when this is turned to High. If you see characters not moving smoothly or flicker, or a background not moving as a whole, experiment a little bit with this setting, and re-save the snapshot when you've found the best setting. (For a slightly more detailed discussion of this option see section 5.1)

J: joystick emulation specifies which Spectrum joystick the PC cursor keys (and analogue or digital joystick, if it is available) control. You can choose from Cursor (default), Kempston, Interface 1 and 2. As I already said above, if Cursor joystick is chosen, the Num-Lock key controls whether Shift is pressed too if the PC cursor keys are pressed. (Since the shift and number keys are pressed exactly simultaneously, it is possible that the Spectrum has already read the Shift key, but not yet the others, when you press both keys down. Sometimes you will therefore get the number 5,6,7 or 8 instead of a cursor movement.)

C: Change speed lets you control the speed of the emulator. As a side effect, slowing down the emulator makes the timing of the various opcodes correspond more exactly to the actual timing on a real processor.

That concludes the discussion of the F4-'change settings' menu. Let's continue with the other function keys.
F5 generates an NMI. Only useful if in SamRam mode; otherwise it may reset the Spectrum or (sometimes) crash a program. ALT-F5 or CTRL-F5 resets the Spectrum.

F6 turns on Real Mode. Try this when the emulator is playing a tune and sounds a little harsh. This mode is needed when you want to load speed-saved games from tape; see below for more information.
F7 and F8 activate the tape- and microdrive-menus. Again, see below for more information.
Resetting the Spectrum, or generating an NMI can be done from the main menu too, in the X - Extra Functions menu. This is useful if you want to activate the NMI software of the SamRam for instance just after loading a snapshot file, or just after you changed the hardware mode. From this menu you can also shell to DOS, and save or load a screen snapshot: a 6912 byte file with extension .SCR that contains a dump of the screen information. This enables you to very easily transfer screens from one Spectrum program to another. The .SCR files can be converted to .GIF or .PCX files by the CONVERT program, available to registered users.

When you're typing BASIC-programs in 48K mode, you'll probably have to look up some Spectrum keywords. Further down in this documentation there is an alphabetical list of all keywords and their key-combination. For 'on-line' help, press ALT-F1 to see the Spectrum keyboard layout.

2.3 Emulation of the Keyboard, Screen and Beeper

The keyboard. Letter keys are mapped to the Spectrum's letter keys. The ALT and CTRL keys can both be used for Symbol Shift. Then, there are a lot of keys on the PC keyboard which don't exist on the Spectrum keyboard. Many of them are used, to make things easier:

- The function keys have several special functions. See the previous paragraph.
- CTRL-Break and CTRL-ALT-DEL quit the emulator.
- The punctuation keys – = ; ' , . / and their shifts: _ + : " < > ? have the effect of pressing Symbol Shift and the corresponding letter key, so you can use these in the straightforward way.
- The ESC key presses Shift-1, EDIT, used as a sort of ESC key in many Spectrum programs. The Backspace key presses Shift-0, the Delete of the Spectrum. CapsLock presses Shift-2, Spectrum's capslock key.
- The PC-cursor keys and the numeric keypad keys 8,4,6 and 2 control the Cursor, Interface 2 or Kempston joystick. The TAB key, and 0,5 and ./DEL on the numeric keypad control the fire button. If the Cursor joystick is selected, you can select whether Shift should also be pressed with the NumLock key (but see the discussion above of the -xs switch).

If you're running the emulator on a slow computer, try selecting double interrupt frequency. Most programs poll the keyboard by interrupt, in any case the ROM does, and doubling the frequency with which this happens will make the emulated Spectrum react much more quickly on your keystrokes.

If you've got an AZERTY keyboard, the standard mappings of the keys won't work at all properly. Include the switch -xz in your Z80.INI file in this case; many punctuation keys will now also work properly. There is no support for other non-US keyboard layouts; sorry!

Now about the screen emulation. Fifty times an (emulated) second, the screen is checked for changes. If anything has changed, the change is displayed on the PC screen. It turned out that this was the fastest method of updating the screen.

I tried to update the screen at about the same time the real Spectrum shows it on the TV screen, relative to the 50 Hz interrupt. There is a problem; the Spectrum takes about 1/100th of a second to generate the whole picture, while I stop the emulator at some point in the 1/50th- of-a-second cycle and display the whole screen at once. Usually this makes little difference, but with some programs it does: characters may flicker heavily or disappear entirely (see for instance BC's Quest for Tires). By selecting the 'video synchronisation mode', you have some control over the exact point of the cycle at which the screen is updated.

In the Hercules, CGA and Plantronics modes, not all colours can be displayed. In the EGA mode, all colours can be displayed, but some colours have the same intensity in bright 1 as in bright 0. In VGA mode, all colours closely resemble the original Spectrum colours, and furthermore in this mode the screen updating is the fastest of all modes.

The border updated every 1/50th of a second, so you cannot see the familiar stripes when saving. However, in real mode the emulator uses the overscan of EGA to display the border, and you can see some stripes there, and in VGA mode the border can be shown full-size. The only drawback of the border emulation in real mode is that there appears some 'snow' on the screen at each OUT - I don't know a way around this.

Finally, the sound emulation. The Spectrum beeper is emulated by the PC's internal beeper. Because every 1/50th of a second the screen has to be updated, and this takes a little time even if there are no changes, the sound is a bit harsh. If you select real mode, the emulator won't update the screen anymore and the sound will sound better.

The sound of the Spectrum 128's sound chip is played through the Adlib card; if you haven't got such a card some notes are played through the internal speaker. That sound will be turned off, however, as soon as the program makes a sound through the normal speaker of the Spectrum. Some

Spectrum 128 programs use the sound chip and the beeper at the same time, and this won't work properly without an Adlib card.

2.4 Using the tape

This emulator can load programs that are saved to tape in the usual way, but also speed-saved programs can be loaded. Furthermore, you can also make a disk file act as an 'emulated tape', so that the normal SAVE and LOAD commands can be used to transfer data to and from disk easily.

Let's first discuss the saving and loading of programs using a tape recorder—that'll be the first thing you want to do, to transfer your programs to the PC. First of all, you need an interface to connect the tape recorder to the PC. The parallel printer interface is used for this. All you need is a very simple and cheap piece of electronics to get the input and output signals at the appropriate and safe levels; the circuit diagram is in the program DIAGRAM.Z80. The interface has to be calibrated; a program to help doing this is contained in the snapshot file. Adjust the variable resistor so that when the tape recorder is played at normal volume, the bar points just below 50%. When the tape recorder is turned off, the bar should go to 0%.

You have to tell the emulator which LPT port you use for tape I/O. This can be selected in the tape menu, but it can also be specified on the command line or in the Z80.INI file with the -b switch; for instance -b2 selects LPT2. Default is LPT1.

There are two ways to load programs: in 'real' or normal mode. In real mode, the emulator doesn't update the screen or scan the keyboard anymore, so that the emulated Spectrum program can run smoothly. The emulator has to run at about 100%, but then you're able to load everything a normal Spectrum would load, including speed-saved programs. The only thing you see on screen are the loading bars in the border (on EGA or VGA screens). Real mode is selected by pressing F6. Saving programs in real mode is a bit useless but it works; enter the SAVE command, press a key to start saving and quickly press F6 when the saving starts. It will continue in real mode.

If your computer is just fast enough, don't slow the emulator down too much. Because the IN instruction is relatively slow, the emulator has to run at about 110% for the best results. If your computer is really fast, you can best slow it down to exactly 100%. If your computer is just a bit too slow, you can try to make your tape recorder run slower too (usually you can do this by adjusting a little screw at the back of the motor), I successfully loaded several speed-saved programs at 90%.

In normal mode, the standard ROM loading and saving routines are 'trapped' (at addresses 04D8 and 056A) when they're about to start saving or loading. A routine in the emulator itself then takes over, and loads or saves a block to tape or a disk file. By default, this routine uses the tape instead of a file, and I'll discuss that mode of operation first.

Using these SAVE and LOAD routines has a great advantage as well as a disadvantage compared to using the Spectrum's own routines in real mode. The advantage is that the internal routines work on every machine, no matter how slow or fast, without having to make the emulator run at 100%. The disadvantage at using them is that they obviously won't understand speed-saved files. For normal use, these internal routines work much easier, and real mode loading is only necessary for speed-saved and very well protected programs.

So far for the general information about tape loading.

The emulator uses files with the extension .TAP to hold a piece of 'tape', with several blocks on it. Each block is usually either a header or a data block; a normal file thus consists of two blocks. There are two modes of operation when loading and saving to disk files, single and multiple .TAP file mode.

In single .TAP file mode, each block saved is appended to the end of the .TAP file, like would happen if you were actually saving to tape. In the same way, when loading in single file mode, each time the ROM wants to load a block, it is presented the next block in the .TAP file. It is handled as it would if the block was loaded from tape, that is, if the ROM needs a header and is presented a data block, it will skip it. The header will however be considered to be read. So, entering `LOAD "rubbish"` will show all headers in the .TAP file, just as an ordinary Spectrum would show all headers on the tape if you left the tape running.

If the last block is loaded, the file pointer is moved to the start again. So a .TAP file can be considered to be an infinite tape. Single .TAP file mode is useful to save whole programs to disk, or for multi-load games that need to load in levels as you play.

A sort of 'random access' file management would also be useful, for instance when you're developing a program and need to save several pieces of data to disk and later load back a specific one. This can be done in single .TAP file mode (by positioning the file pointer using the Browse function), but there's a different mode of operation that makes things easier: multiple .TAP file mode. In fact, by default the emulator is in this mode.

When the emulator is in multiple .TAP file mode, it will read all blocks from all .TAP files in a specified directory, one after the other. When it has finished reading the last one, it will start all over again.

When saving, the emulator will put the two blocks of a normal file, the header and the data block, in one .TAP file with a unique name made up of the printable letters of the file name and a two-digit number. The name of the .TAP file is irrelevant to the emulator, but to have it resemble the name of the actual Spectrum file you saved is simply convenient. If the Spectrum program saves a data block to tape without first saving a header, the .TAP file will contain only this data block, and the DOS file name will be HDRLES, with a two-ditit number appended to make it unique. The format of the .TAP files saved in multiple .TAP file mode is exactly the same as the format used in single .TAP file mode.

You can easily string together .TAP files; for instance a number of .TAP files created in multiple .TAP file mode can be put into one big .TAP file simply by copying them together, e.g.

```
COPY /B FILE1.TAP + FILE2.TAP ALL.TAP
```

(Note: in some versions of DR DOS the /B switch, necessary because otherwise copying stops after a CTRL-Z character, doesn't work properly; load your old COMMAND.COM to copy the files).

Now you know what you can do, but how to get the emulator to do it? That's what the final section is about: the tape menu.

Press F7 to enter the tape menu. Pressing S will select or de-select single file mode. By default, multiple .TAP file mode is selected. In this case, there are three other possible choices in this menu. First of all, D selects a tape-file directory where the .TAP files will be saved into and loaded from. A default directory can be selected by putting the -xs switch on the command line or in the Z80.INI file; for example -xs c:\spectrum\taps selects that directory.

The I and O options are used to select the source and destination of the saving and loading: the LPT port for a physical tape recorder, or 'disk' for disk files. By default LPT1 is selected; another LPT port can be selected with for instance -b2 or by pressing I and O. Input and output are directed to disk by default if a default tape file directory is given by means of a switch on the command line or .INI file.

If Single .TAP file mode is selected, different and more menu options appear. With G and P, the input and output tape files can be selected. They may be the same. If a specified output file already exists, you may choose to append to or overwrite this old file. Saving is always at the end of the file; loading always starts at the beginning of the .TAP file.

With the B option - Browse - the position of the file pointer into the input .TAP file can be changed. If you, for instance, type `LOAD""` instead of `LOAD "" CODE`, the first header is read, and you would have to read all other headers before trying to load the file again. With the browse option you can conveniently change the file pointer. Of every header (that is, every block with flag byte 0 and length exactly 17) the name and type, and of every data block the length is shown.

The option B can also be used to delete specific blocks from a .TAP file. Make sure you do not only delete a data block or a header, or the ROM may get confused! (Double data blocks will be skipped, but double headers can generate Tape Loading errors).

As in multiple .TAP file mode, I and O are used to specify the source and destination for saving and loading. If you enter a .TAP file name with G or P, this will automatically be set correctly. You can then always reset the input or output back to LPTn again, of course.

Finally, in Single .TAP file mode you can use 'tape mirroring': loading programs from tape (in normal mode, i.e. not using Real mode) and at the same time saving a copy of each block loaded into a .TAP file. This .TAP file can later be used to load the program again, might anything go wrong. There are two ways of mirroring: normal mirroring and exact mirroring. The last one must be used only in exceptional cases; it will always make a copy of a block, even if it had a tape error (the corresponding block in the .TAP file will also have a tape error). This causes ticks in leader tones to make 0-byte blocks, so the .TAP file may get messy. Do not use exact mirroring if you don't really have to; I think normal mirroring will always work in practice.

If you try to leave the tape menu when for instance tape mirroring is selected, and no output filename is given, the emulator will warn you and will insist that the error be corrected. Yes, it's stubborn!

2.5 Using the microdrive

Compared to the tape, this is really simple. Cartridges are emulated by files of 137923 bytes. These files have the extension .MDR, and can contain up to 126K of data. The emulator emulates 8 microdrives, the maximum amount the Interface I software can handle, and each of these cartridge files can be inserted in any of the 8 microdrives. (Do not insert one file into more than one microdrive; this will cause problems with the buffering done by the emulator as well as the Interface I, and might result in data loss).

Press F8 to enter the microdrive menu. Press 1 to 8 to select a microdrive, and I to insert a microdrive cartridge. You can select an existing one, or type a new name. If the cartridge file isn't found, the emulator asks whether it should create it. When created, you'll have to format it first; if you don't, you'll get a 'microdrive not present' error when you try to read it, just as happens with real unformatted cartridges. To format a cartridge, type

```
FORMAT "m";1;"name"
```

After this the cartridge should have 126K of free space.

The cartridge can be write protected; see the menu option in the F8 menu. This is a characteristic of the cartridge, and the write protect tab information is therefore stored in the cartridge file.

As on the real Spectrum, you'll have to be careful with OUT's if a cartridge is inserted. Try `OUT 239,0` (on a real Spectrum, this turns on the microdrive motor) and wait a few seconds; most of your data will be lost! You can stop the microdrive motor by typing STOP (or, more generally, generate an error).

The microdrives are emulated at IN/OUT level. This means that every utility or program that uses microdrives ought to work on the emulator. Most utilities use hook codes, and these will certainly work.

The GAP line is emulated; this signal is activated if the interface I senses a piece of tape with no data on it. If the checksum of the first header block of a microdrive header or data block is not correct, that block is considered to be a GAP. This will only happen if some utility writes a bad block to microdrive deliberately, if the file is newly created and unformatted, or when you type `OUT 239,0`.

2.6 Using the RS232 channel

This was the only Spectrum i/o channel that could be used in the early versions of the emulator. Using .TAP files instead of the RS232 channel is often easier, but sometimes using the RS232 channel can be very useful too, for instance if you've got a null-modem lead that connects a Spectrum with interface I to the PC you can use it to transfer data and programs easily. Furthermore, the RS232 channel is the easiest way to let the emulator communicate with a PC printer.

The Interface I RS232 port is called the "B" or "T" channel. The first is the binary channel, the "T" channel won't let all control codes through and will expand any keyword; useful for LISTing a program but otherwise annoying.

The Spectrum 128 has its own RS232 port; it is called the "P" channel. Output to either the Interface I's or Spectrum 128's own RS232 port will all be processed as 'RS232 output', and input will go to both (that is, to the one you happen to read from).

The output to the RS232 channel can be routed to an LPT port, to a COM port or to a file on disk. Input can come from either a file or a COM port.

If you want to use the RS232 channel for printing using `LPRINT` and `LLIST` (shorthand for `PRINT #3` and `LIST #3`), be sure to open that channel for output to RS232; by default it sends its output to the ZX Printer, which is not supported. You can open the channel by typing `OPEN #3,"B"` (or "T" for listings, or "P" on a Spectrum 128).

Input and output are buffered. This is important to remember when you're transferring files using the `SAVE` and `LOAD *"b"` commands of the Interface I. If the header is missed, for instance if you try to load the wrong file type, re-sending the file will not directly work because there will still be bytes in the buffer. You have to clear the input buffer before re-sending the file. When inputting from a disk file, the file pointer can be reset to point to the start of the file again to re-read the header.

When inputting or outputting from or to a disk file, the read or write position is displayed as a byte-count. An <EOF> sign will appear if an input file is read completely through to the end.

The RS232 redirection options are in the Change Settings (F4) menu. The menu options are pretty obvious if you keep above remarks in mind, so I won't go into that.

When using a COM port, make sure you have initialised it before starting the emulator with the Dos MODE command, for instance

```
MODE com1:96,n,8,1
```

initialises COM1 to send and receive at 9600 baud, no parity, 8 data bits and 1 stop bit, the default for the Interface I.

Here is how to transfer programs from a Spectrum to the PC using the RS232 lead. First, you need a null-modem lead. I myself use the following cable:

Spectrum (9 pins)			AT (9 pins)	PC (25 pins)
3	TxD	——————— RxD	2	3
4	DSR	——————— DTR	4	20
		CTS	7	4
		|		
		RTS	8	5
7	GND	——————— GND	5	7

(so CTS and RTS have to be connected!) This is not a full null-modem lead; you can only send data from the Spectrum to a PC. Here's how to transfer: load the program SAVESPEC.Z80 in the emulator and type the basic program over into the real Spectrum, and run it. It saves a short piece of code to tape.

Now load the program you want to transfer, and stop it. (This may be tricky!) Load the code back into memory at address 16384 (the code is relocatable but this is the safest place):

```
LOAD "RS232" CODE 16384
```

Now open channel three for output to RS232; on a Spectrum with Interface I this would be `OPEN #3,"b"`, on a Spectrum 128 it would be `OPEN #3,"p"`, and with other interfaces you'll probably know what to do. Select the right baud rate on the Spectrum (probably `FORMAT "b",9600` or something like that). Now initialise the appropriate COM port on the PC and type

`GETRS /n filename.z80` (n=COM port used)

at the DOS prompt, and then type `RANDOMIZE USR 16384` to send the whole memory over to the PC. The resulting .Z80 file should now be exactly 49182 bytes long (that is 48K+30 bytes), if not try again or try a lower baud-rate. Voila, transferred!

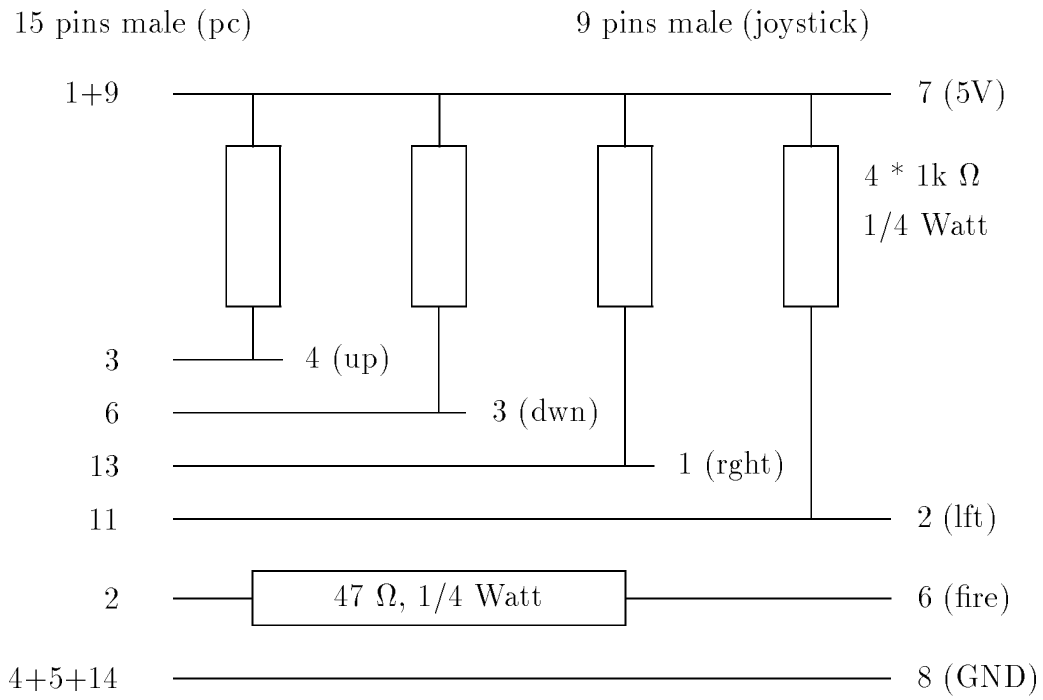
To transfer short blocks of data it's often easier to use the `LOAD *"b"` and `SAVE *"b"` commands of the Interface I. When the right options have been selected in the RS232 i/o redirection menu, you should just follow the instructions of the Interface I user manual and all should work as expected.

2.7 Joysticks

As was already said in the introduction, the emulated Spectrum joystick (Cursor, Interface 2 or Kempston) is controlled by the PC cursor keys and 5/0/. on the numeric keypad and TAB as fire keys. The emulated joystick can also be controlled by a real joystick, both an analogue (PC standard) or a digital one.

The analogue joystick support is rather straightforward. If you've got one, it works - it couldn't be simpler. The digital joystick support is less obvious, since PC's don't support these.

To use digital joysticks, Ruud Zandbergen has made a device that uses the two inputs of a normal analogue joystickinterface to connect a digital joystick to a PC. Here's the circuit diagram:



4+5+14 means: connect pins 4, 5 and 14. The same applies for pins 1 and 9. Here's the list of ingredients:

- 1 x 9 pins D plug, male
- 1 x 15 pins D plug, male
- 4 x 1k Ω , 1/4 Watt resistors
- 1 x 47 Ω, 1/4 Watt resistor
- piece of 7-wire flatcable

Everything can be fit into the 15-pins plug. Make sure the resistors don't touch the other blank connections! This interface can be used for all usual digital joysticks, with or without auto fire (that is every joystick that work with a Kempston joystick interface, or that work on a Commodore 64/Amiga or Atari). The joysticks for the Spectrum +2/+3 will not work, however the pin layout is easy to change.

This joystickinterface needs an analogue PC-joystickinterface on which you can connect TWO analogue joysticks (on one plug!). Most cards can do this, but some multi-I/O cards support only one joystick. Check the documentation of your I/O card to see whether your joystickinterface is suitable. The soundblaster joystick interface works fine.

A number of PC games will behave strange when the digital joystick interface is connected; they run very slow or crash. When this happens, remove the joystick interface (not only the joystick!).

2.8 Transferring programs

There are a number of ways to transfer programs from the Spectrum to the PC: loading them directly from tape, using the RS232 lead or transferring from disks of Spectrum disk interfaces. And then you might have snapshot files from other emulators that you want to convert to .Z80 files. I'll discuss these cases one after the other.

Converting using the COM port is not so easy most of the times, but if you've got a null-modem lead waiting to do something you could read section 2.6. Luckily, there are easier ways.

First of all, you can use the tape. If you want to do this, then the first thing to do is to read section 2.4 carefully - now you know almost everything you need. Most programs you have probably use the normal tape format; you will find that these usually load right away. If the programs use speed-load, using real mode will probably load most of these right away too.

But some programs are really cleverly protected, and use obscure features of the Z80 processor. To run these programs, turn on LDIR emulator and R register emulation (see the 'Change Settings' menu, F4). Note that the emulator will slow down a bit when R register emulation is selected; if you need to use real mode then make sure you speed the emulator up again to 100%. After the program has loaded successfully, you may try to turn R register emulation off again; I don't know any program that needs R register emulation after loading. Read chapter 5 for more technical information about these options.

If you've got Spectrum disks, you will probably be able to convert the programs on them to a useful format and use them in the emulator. The registered package of this emulator contains a program DISCIPLE, that can read DISCiPLE and Plus D disks and convert the snapshots and other files on it to .TAP and .Z80 files. The previous version of this program could only read 3.5" Disciple disks, and had several bugs in the file and snapshot translation routines. So if you transferred programs with the old DISCIPLE program and they don't work, don't blame the emulator but try to transfer them again with the new program.

The current version of the DISCIPLE program reads 3.5" as well as 5.25" DISCiPLE disks, will translate 48K, 128K and screen snapshots, and other normal files. The previous version used the .SAV file format for normal files, which could be loaded into the emulator using LOAD *"b"; this version converts them into .TAP files which can be loaded simply by using the normal tape LOAD statements (see 2.4).

The DISCiPLE interface modifies the Spectrum system variables in such a way that LPRINT sends its output to DISCiPLE's own printer interface. When you transfer a snapshot that uses the printer, you'll have to tell it to use the Interface I's RS232 printer output instead, by breaking the program and typing `OPEN #3,"b"`. If you don't, you'll get strange results.

If you have got a Beta disk interface, your problem is solved too. J.L. Bezemer wrote a program called BDDE that reads Beta disks. The program can be downloaded from the Spectrum emulator support BBS.

Finally, maybe you were using another Spectrum emulator for the PC before using this one, and you may have already got a collection of snapshot or other files. CONVZ80, another utility for registered users, can convert between several snapshot formats, namely the .SNA format of JPP, the .SP formats of VGASPEC and SPECTRUM, the .PRG files of SpecEm, and the .Z80 format of course. (It is by the way not necessary to convert .SNA files, the emulator can read them as they are.) CONVZ80 can also convert the tape files used by SpecEm and ZX to .TAP files. CONVZ80 recognizes what it should do by the extension of the files you enter on the command line; to distinguish between VGASPEC's and SPECTRUM's .SP formats you can use the switch -o. If the extension consists of digits only, it is taken to be a ZX tape file, and if it contains non-digits and is none of .SP, .Z80, .SNA, .PRG or .TAP it is regarded as a SpecEm tape file.

SpecEm can load .PRG snapshot files, but cannot save them. However, it emulates the Multiface I, which can save snapshots to tape. SpecEm will save these blocks as tape files to disk. If you convert these to a .TAP file (in the correct order!), you can load them into Z80 and save the program as a .Z80 file.

2.9 Converting file formats – the utility CONVERT

This section is about the utility CONVERT, which can convert some of the Spectrum's own format into each other, and also converts some of the emulator's formats into others. It is not about converting files from other emulators; read section 2.8 if you want to know about that.

CONVERT was useful when the emulator could only communicate with snapshot files and the RS232 link. It has become less useful now, with .TAP files, but it still has some useful features.

It can read three types of input files: pure ASCII, pure bytes (for instance a .SCR screen dump), and files produced by a `SAVE *"b"` command.

Output is pure bytes, ASCII with either CR (Spectrum standard) or

CR/LF (PC standard) for line breaks, `SAVE *"b"` files containing a Basic or code file, a .PCX or a .GIF file.

So what can you do? Main uses are adding LF (10 hex) bytes to a text file produced by the Spectrum; converting a code block into a SAVE *"b" to load it into the Spectrum using `LOAD *"b"` (and the reverse of course: converting a `SAVE *"b"` file to pure bytes), and converting a screen dump to .PCX or .GIF graphics files.

Less useful, but possible: LISTing a program (`SAVE *"b"` file) to produce readable ASCII, and the reverse: converting an ASCII listing to executable Basic again.

If you want to make a .PCX or a .GIF file, input should be a `SAVE *"b"` file of a screen (length 6921 bytes exactly) or a bare .SCR screendump (length 6912 bytes). You can make screendumps by selecting the X-Extra functions menu from the main menu.

2.10 The utilities Z802TAP and TAP2TAPE

The SamRam has built in it some snapshot software. Using this software you can save any 48K Spectrum program to tape or to a .TAP file, as is explained in section 3.2 below. But the SamRam software cannot handle a 128K program.

The utility that can convert a 128K snapshot (and 48K ones for that matter) to a .TAP file is called Z802TAP. The .TAP file includes a basic loader, and a loading screen if you want. Z802TAP compresses the blocks it writes (using a better method than used in compressing .Z80 files) to save loading time. If you don't want it to compress the blocks, for instance when you want to take a look at the ram pages of the Spectrum 128, specify -u when you run Z802TAP. You can load the converted program simply by executing

```
Z80 -ti tapefile
```

and typing `LOAD ""` (for a 48K program) or changing the hardware mode to Spectrum 128 and choose 'Tape Loader' in the menu.

The program TAP2TAPE writes .TAP files back to tape. The program consists of a batch file TAP2TAPE.BAT, which executes the TAP2TAPE.Z80 file using the emulator. The .TAP file is written to tape exactly as it is, so that if a block contains a tape error, it won't load correctly from tape either. If the entire .TAP file has been saved the emulator will start loading from tape. At that point, press space once to return to DOS.

3 THE SAMRAM

3.1 Basic extensions

The SamRam is a hardware device Johan and I built for our Spectrums. It consists of a 32K static RAM chip which contains a modified copy of the normal Basic ROM and a number of other useful routines, like a monitor and snapshot software. You can compare it to a Multiface I interface, but it's more versatile. Another useful feature was a simple hardware switch which allowed use of the shadow 32K Ram, present at 8000-FFFF in most Spectrums, but hardly ever actually used.

For more details on the low-level hardware features of the SamRam read chapter 5. In this chapter I'll explain the software features of the SamRam software, somewhat bombastically called the 'SamRam 32 Software System' or the 'Sam Operating System'. By the way, all similarity between existing computers is in fact purely coincidental and has in no way been intended. Really!

The SamRam offers a few new Basic commands, and a lot of useful routines that are activated by an NMI, i.e. by pressing F5. First I'll discuss the Basic extension.

Select the SamRam by starting the emulator with the -s switch, or by selecting it from the F9 menu. Normal Basic functions as usual; the character set is different from the original one. There are four new commands:

`*RS`, `*MOVE`, `*SAVE` and `*SPECTRUM`,

and two new functions, `DEC` and `HEX`, which have replaced `ASN` and `ACS`. DEC takes a string argument containing a hexadecimal number, and returns the decimal value of it. `HEX` is the inverse of the `DEC` function, and yields a four-character string.

- `*RS` sends its arguments directly to the RS232 channel. You don't have to open a "b" or "t" channel first. You're right, it's of limited use. Example: `*RS 13,10`

- `*MOVE` is useful: it moves a block of memory to another place. Example: `*MOVE 50000,16384,6912` moves a screen-sized block from 50000 to the start of the screen memory.

- `*SAVE` works like `*MOVE`, except that it activates the shadow SamRam ROM before moving. I used this command to update the shadow ROM, but on the emulator you can use it to move the shadow ROM to a convenient place in Ram where you can take a look at it, for instance by executing `*SAVE 0,32768,16384`.

- `*SPECTRUM` resets the SamRam Spectrum to a normal one. You lose all data in memory. By resetting the emulator by pressing ALT-F5, the SamRam is activated again. Not very useful either.

- Then there's the Ramdisk, which is, like the Spectrum 128 ramdisk, accessed via the `SAVE!`, `LOAD!`, `CAT!`, `ERASE!` and `FORMAT!`. The syntax is straightforward. `FORMAT!` and `CAT!` need no parameters; `ERASE!` only needs a name. If a file is not found, the SamRam will respond with a 5-End of File error. The Ramdisk has a capacity of 25K.

3.2 The NMI software

Select the SamRam (F9-3), and press F5. A menu with eight icons pops up. You can select each icon by moving the arrow to it (using the cursor keys or the Kempston joystick), and pressing '0' or fire. The icons can also be selected by pressing the appropriate letter key.

The eight icons are two arrows with N and E within them, a magnifying glass with the letters 'mc' in it (activated by pressing D), two screens (identified by 1 and 2), a printer (P), a cassette (S) and a box saying 'overig'. The 'D' activates the monitor or disassembler; read section 3.3 for information on this program.

Pressing N or E returns you to the Spectrum. If you pressed N, the normal Spectrum rom will be selected when the NMI software returns; if you press E, the Rom with the Basic extensions will be selected. Some games may crash if they see a different rom than the standard Spectrum one.

Pressing 1 selects the tiny screen editor. You can move a '+' shaped cursor about the screen using the cursor keys. The following commands are available:

H: Get the current ATTR color from the screen at the cursor's current position, and store it in memory. This color will be used by the next command:

Z: Put the color on the screen

G: Get a character from the screen

P: Put the character on the screen

R: Remove all screen data that is invisible by the ATTR color

L: Take a look at the bitmap below the ATTR color codes

T: Return to the main menu. You can also return by pressing EDIT, or ESC in the emulator.

B: Change border color

V: Clear the whole screen

If you press 0, you can edit the current 8x8 character block at pixel level. Again you control the cursor with the cursor keys. Now 0 toggles a pixel. In this mode there are two commands: C clears the whole block, and I inverts it. Pressing EDIT (ESC) returns you to the big screen again.

The SamRam has two screen buffers. Buffer 1 is used to hold the screen which was visible when you pressed NMI, to be able to restore it when returning. This is the screen you edit with '1'. The second screen buffer can be used to hold a screen for some time; it is not touched by the NMI software directly, and will not even be destroyed by a Reset. If you press '2', a menu appears with four Dutch entries:

1: Scherm 1 opslaan (Store screen 1 into buffer 2)

2: Scherm 2 veranderen (Edit screen 2)

3: Schermen verwisselen (Swap screens)

4: Scherm 2 weghalen (Remove screen 2)

These four functions are rather obvious, I believe.

Pressing 'P' pops up the printer menu. The screendump program is written specifically for my printer, a Star SG-10. It will probably work on some other printers, but not on most. The output is sent to the RS232 channel, so you have to redirect it to an LPT output.

Skipping the most interesting, 'S', for a moment, let's first discuss the final menu, 'O' for 'Overig', Dutch for miscellaneous. There are five menu options, of which three are not useful. The first gives a directory of the cartridge currently in Microdrive 1. The last, 'E', returns you to Basic if this is anywhere possible: it resets some crucial system variables and generates a Break into Program. You can use this for instance to break in a `BEEP`, or crack a not-so-very-well-protected program. The three other options select normal or speed-save, and store the current setting in CMOS Ram. Speed-save won't work properly on the emulator, because the speed-save routine toggles the upper 32K ram bank regularly, and this takes too much time on the emulator. The setting is not important if you use the internal save routine (which will be used by default, unless you select Real Mode).

Finally, the 'S' option. This option allows you to save a snapshot to tape or microdrive. I used it a lot on my real Spectrum, and it works just as well on the emulator. It is very useful is you want to load a .Z80 program back into a real Spectrum again. There are three 'switches' you can toggle.

The active choice is indicated by a bright green box, inactive boxes are non-bright. You have to use EGA or VGA to be able to see it... The first switch lets you select whether the SamRam rom should be active if the program loads or not. This is only meaningful is you load it back in a SamRam again. Usually I want the SamRam rom to be active because I like the character set better. The second switch indicates whether the SamRam should save a 'loading screen', which it takes from screen buffer 2. If screen buffer 2 contains a screen, this switch will by default be on. Finally, the last switch lets you select the output media, tape or cartridge.

If the program is loaded back into the SamRam, the only bytes that have been corrupted are four bytes down on the stack; this will virtually never be any problem. If the program is loaded back to a normal Spectrum, these four bytes will also be corrupted, and the bottom two pixel lines of the screen will be filled with data. (This is considerably less than any other snapshotter I've seen: for instance the Multiface I uses more than 35% of the screen!)

The Microdrive BASIC loader needs code in the SamRam rom to start the program (the `RANDOMIZE USR 43` calls it). It won't be very difficult to write a standard BASIC loader that doesn't need this code, but I don't think many people desperately need it...

3.3 The built-in monitor

This is a really very convenient part of the emulator, and I use it a lot. It is very MONS-like in its commands and visual appearance. It cannot single-step however, but on the positive side it has some features MONS hasn't. It is a part of the SamRam, and cannot therefore be used with Spectrum 128 programs. If you want to take a look at a Spectrum 128 program, press F10, then change the hardware to SamRam without resetting, and finally generate an NMI in the Extra Functions menu. You won't probably be able to continue to run the program, but at least you're able to see what it was doing.

Press F5 for NMI, and D to enter the monitor/disassembler. The first eight lines are the first eight instructions, starting at the Memory Pointer, from here on abbreviated by MP. At first, MP is zero. The disassembler knows all official instructions, and the SLL instruction. If another inofficial instruction (i.e. starting with DD, FD or ED) is encountered, the first byte is displayed on a blank line. The four lines below these display the value of PC and SP, the first nine words on the stack (including AF and the program counter, which have been pushed during NMI), and three MP-memories. These can be used for temporary storage of the MP, for instance when you take a look at the body of a CALL, and want to return to the main procedure later.

The bottom part of the screen displays 24 bytes around the memory pointer.

Commands are one letter long; no ENTER needs to be given. If one or more operands are needed, a colon will appear. By default the monitor accepts hexadecimal input. A leading $ denotes that the number is to be regarded as decimal. If you give the # command, the default will toggle to decimal, and you need to explicitly put a # in front of a number which is to be interpreted as a hex number. Also, after the # command all addresses on screen will be decimal. A single character preceded by the " symbol evaluates to its ASCII code, and the single character M will evaluate to the current value of the memory pointer. The monitor commands:

Q: Decrease the memory pointer by one. You effectively shift one byte up.

A: Increase the memory pointer, shifting one byte down.

ENTER: Shift one instruction down: the memory pointer is increased by the length of first instruction displayed on screen.

M: Change the value of the memory pointer. For instance, M:M won't change it.

P: Put. The word operand supplied will be stored in the first MP memory, and the others will shift on place to the right. Usually, you'll want to store the memory pointer by P:M

G: Get. Typing G:1, G:2 or G:3 moves the value of one of the MP memories to the MP.

B: Byte. This command needs a byte operand; it will be poked into memory, and the memory pointer will move one up.

I: Insert. The same as B, except that you can poke more than one byte. It continues to ask for bytes to poke until you type Enter on a blank line.

#: Toggles the default number base between hexadecimal and decimal.

F: Find. You can enter up to ten bytes, which will be searched through memory. Searching will stop at address 0, because since the search string is stored in shadow Ram, searching would otherwise not always terminate. Typing Enter on a blank line starts the search. Byte operands are entered as usual, but:

- If a number bigger than 256 decimal is entered, it is treated as a word in the standard LSB/MSB format. So, 1234 will search for 34,12 hex in that order. Note that 0012 will search for 12, not 12,00.
- A line starting with " decodes into the string of characters (up to ten) behind it. Normally this would only be the first character. So instead of typing "M "Y "N "A "M "E (space=enter here) you type "MYNAME. Note that any terminating " will also be searched for!
- An x is treated as a wildcard. So if you search for CD x 80 any call to a subroutine in the block 8000-80FF is a hit. If you search for x 8000, you'll see every one-byte instruction that has the address 8000 as operand.

N: Continues the search started by F from the current MP.

$: Displays one page of disassembly on screen. In this mode, the following commands are possible:

$: Back to the main screen

7: [Shift 7 also works, cursor up]: Go to the previous page. The monitor stores the addresses of the previous eight pages only.

Q: Go back one byte (decrease MP by one)

A: Go one byte forward (increase MP by one)

Z: Dump this screen to the printer, in ASCII format. Redirect the RS232 output to a file, and run CONVERT on it to convert the CR's into CR/LF's before printing (or tell your printer to do the conversion).

Every other key displays the next page of disassembly.

K: List. The same mode as with $ is entered, but instead of a disassembly the bytes with their ASCII characters are displayed. Useful to look for text.

C: Clear. Fills blocks of memory with a specified value. The monitor prompts with 'First', 'Last' and 'With'. The 'Last' address is inclusive!

D: Dump. Prompts with 'First' and 'Last', and dumps a disassembly of the block between these addresses to the printer. See remark at $-Z. The 'Last' address is again inclusive.

- R: Registers. If you press Enter after R, an overview of the registers contents is displayed. If you type one of A, B, C, D, E, H, L, A', B', C', D', E', H', L', I, R, AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, SP or PC, you can change the value of it. Changing the value of SP also changes the PC and AF values by the way. You cannot change the Interrupt mode or IFF.
- V: Verplaats (Move). Prompts with 'From', 'To' and 'Length'. Obvious.
- S: Save. Enter the start of the block you wish to save first. The monitor then prompts with 'Length'. The block is saved without a header, as a normal data block (A, the flagbyte, is 0FF)
- L: Load. Loads a block of data from tape, at the specified address. Normal data blocks, headers and blocks with non- standard flag bytes can be loaded. The first byte in memory will contain the flag byte. If the checksum isn't 0 after loading, indicating a tape error, you'll hear a beep.
- H: Header read. Loads headers and displays the contents on screen.

As you're reading this part, I assume you know something of machine code. Probably you would be interested in peeking into the software of the SamRam, the Interface I or the Spectrum 128. You'll first have to move these roms in ram to be able to look at them with the monitor.

The Interface I rom can be moved into ram by saving it to microdrive or to the "b" channel, with:
`SAVE *"m";1;"rom" CODE 0,8192` or `SAVE *"b" CODE 0,8192`, and loading it back again at 32768 for instance. You can also put this small machine code routine at 23296 and run it: F3 21 0C 5B E5 21 00 00 E5 C3 08 00 21 00 00 11 00 80 01 00 20 ED B0 FB C3 00 07.

The two SamRam roms are easy. The first you don't need to transfer; the monitor looks at the extended basic rom by default. The second rom can be moved to 32768 by typing `*SAVE 0,32768,16384`. (The SAVE is not the keyword SAVE!)

The first '128 rom, the one which is active at reset and contains most of the new code, is moved up by typing `SAVE!"rom"CODE 0,16384`, then `LOAD!"rom"CODE 32768`. The other rom is most conveniently moved by saving it to a .TAP file and loading it back again in ram. To select the SamRam type SPECTRUM first, and then switch the hardware without resetting.

4 THE SPECTRUM

4.1 The Spectrum

This emulator supports the Interface I and the Spectrum 128. Many Spectrum users will have no experience with them, so some comments may be useful. On the other hand, I don't think this is the right place to describe the Spectrum Basic in full detail. If you want to know it all, read the official manuals!

If you want to use Spectrum Basic, you will need the keywords. You could by the way now also use the Spectrum 128 Basic where you can type the keywords in by full.

If you press ALT-F1 in the emulator, the Spectrum keyboard layout will appear. For completeness I include an alphabetical list of all keywords and their key-combination. In the list below, K stands for Keyword mode, E for E-mode (type Shift-Alt of Shift-Ctrl to select E-mode), S for Symbol Shift, and SE for Symbol Shifted (Alt/Ctrl) E-mode: select E mode and type the letter while depressing Symbol Shift.

Character	Spectrum-Keyb.	PC-Keyboard
&	S 6	ALT (or CTRL) 6
’	S 7	ALT 7 or ’/”
(	S 8	ALT 8
)	S 9	ALT 9
--	S 0	ALT 0 or SHIFT --/-
<	S r	ALT r or SHIFT </,
>	S t	ALT t or SHIFT >/,
;	S o	ALT o or :/;
”	S p	ALT p or SHIFT ”/’
ˆ	S h	ALT h
-	S j	ALT j or _/-
+	S k	ALT k or SHIFT +/= oder GREY +
=	S l	ALT l or +/=
:	S z	ALT z or SHIFT :/;
?	S c	ALT c or SHIFT ?//
/	S v	ALT v or ?//
∗	S b	ALT b or GREY PRTSC/*
,	S n	ALT n or </,
.	S m	ALT m or >/.

Keyword	Code	Keyword	Code	Keyword	Code
ABS	E g	GO TO	K g	PRINT	K p
ACS	SE w	IF	K u	RANDOMIZE	K t
AND	S y	IN	SE i	READ	E a
ASN	SE q	INK	SE x	REM	K e
AT	S i	INKEY$	E n	RESTORE	E s
ATN	SE e	INPUT	K i	RETURN	K y
ATTR	SE l	INT	E r	RND	E t
BEEP	SE z	INVERSE	SE m	RUN	K r
BIN	E b	LEN	E k	RAVE	K s
BORDER	K b	LET	K l	SCREEN$	SE k
BRIGHT	SE b	LIST	K k	SGN	E f
CAT	SE 9	LINE	SE 3	SIN	E q
CHR$	E u	LLIST	E v	SQR	E h
CIRCLE	SE h	LN	E z	STEP	S d
CLEAR	K x	LOAD	K j	STOP	S a
CLOSE #	SE 5	LPRINT	E c	STR$	E y
CLS	K v	MERGE	SE t	TAB	E p
CODE	E i	MOVE	SE 6	TAN	E e
CONTINUE	K c	NEW	K a	THEN	S g
COPY	K z	NEXT	K n	TO	S f
COS	E w	NOT	S s	USR	E l
DATA	E d	OPEN #	SE 4	VAL	E j
DEF FN	SE 1	OR	S u	VAL$	SE j
DIM	K d	OUT	SE o	VERIFY	SE r
DRAW	K w	OVER	SE n	<=	S q
ERASE	SE 7	PAPER	SE c	>=	S e
EXP	E x	PAUSE	K m	<>	S w
FLASH	SE v	PEEK	E o		
FN	SE 2	PI	E m		
FOR	K f	PLOT	K q		
FORMAT	SE 0	POINT	SE 8	DEC	SE q
GO SUB	K h	POKE	K o	HEX	SE w

4.2 The Interface I

If you want to use the microdrive, you'll need cartridge files. The emulator can create an empty cartridge file for you. You have to format it before you can use it. Type

```
FORMAT "m";1;"name"
```

to format the cartridge currently in Microdrive 1 giving it the name 'name'. Next, type `CAT 1` to get a catalogue of the files on it (none of course) and the number of kilobytes free. You can save a file by typing for instance

```
SAVE *"m";1;"screen"SCREEN$
```

Instead of `SCREEN$` you can use all other expressions that are permitted also when saving to tape, like `LINE nnnn` or `CODE x,y` etcetera. To load a file back from cartridge, you type (you guessed it)

```
LOAD *"m";1;"screen"SCREEN$
```

If the file doesn't exist or is of the wrong type you'll get the appropriate error message. To erase a file, type for instance

```
ERASE "m";1;"screen"
```

Note that no * is needed (or even permitted), and that only the name should be given. There's another way to create a file on a cartridge, and that is by using a command like `OPEN #3;"m";1;"name"`, and printing to that stream. You can use `MOVE` to move data from stream to stream, but I'll not go into that — it's not very much used anyway.

Instead of to the microdrive, you can also 'save to the RS232 link'. For instance, type `SAVE *"b"SCREEN$` (note: there's no name!) to save a screen. On the emulator you can send the output to the RS232 channel to a printer (then `SAVE *"b"` is useless), to a file (can be useful) or to the COM port (very useful if you connect a real Spectrum to the PC's COM port!). You can load the data back by typing `LOAD *"b"SCREEN$` and making sure the RS232 channel is fed with the right input (from a COM port or a file). See also paragraph 2.6.

If you want to use the RS232 channel for printing, open stream 3 for output to that channel by typing

```
OPEN #3,"b"
```

or

```
OPEN #3,"t"
```

The first will simply copy everything you send to stream 3 (using for instance `LPRINT` or `LLIST`) to the RS232 channel; the second converts CR's into CR/LF's, breaks off lines at 80 characters and translates keywords into character sequences. "t" is useful for LLISTings, but not for anything else.

Useful extra commands: `CLS #`, to clear the screen and reset the attributes to their reset defaults, and `CLEAR #` to do a `CLS #` and close all currently open streams (discarding all data that may still be buffered!)

The Interface I uses its own system variables. At the first error message you make (or RASP, or flashing question mark) and at the first Interface I statement you execute, it inserts them automatically. Some programs will not run when the Interface I has inserted its system variables. So if you load a game from tape, reset the Spectrum first and don't make an error typing `LOAD ""`. With a bit of exercise you should be able to do this.

4.3 The Spectrum 128

The main new features of the Spectrum 128 are its larger memory, that can be used as a Ram drive in Basic, and music capabilities.

The Ram drive is accessed via the `LOAD!`, `SAVE!`, `ERASE!` and `CAT!` commands. They work as you would expect. Examples:

```
SAVE!"name"SCREEN$

CAT!

LOAD!"name"SCREEN$

ERASE!"name"
```

The 3 channel sound chip of the Spectrum 128 can be used in Basic with the `PLAY` command. Example:

```
PLAY "cde","efg","gAB"
```

plays three chords. You can program complex effects, melodies and rhythms with the play command; they require many commands in the three voice strings which I won't explain...They are explained in the Spectrum 128's user guide.

5 TECHNICAL INFORMATION

5.1 The Spectrum

The Spectrum is at the hardware level a very simple machine. There's the 16K ROM which occupies the lowest part of the address space, and 48K of RAM which fills up the rest. An ULA which reads the lowest 6912 bytes of RAM to display the screen, and contains the logic for just one I/O port completes the machine, from a software point of view at least.

Every even I/O address will address the ULA, but to avoid problems with other I/O devices only port FE should be used. If this port is written to, bits have the following meaning:

Bit	7	6	5	4	3	2	1	0
				E	M	Border		

The lowest three bits specify the border colour; a zero in bit 3 activates the MIC output, and a one in bit 4 activates the EAR output (which sounds the internal speaker). The real Spectrum also activates the MIC when the ear is written to; the emulator doesn't. This is no problem; MIC is only used for saving, and when saving the Spectrum never sounds the internal speaker. The upper three bits are unused.

If port FE is read from, the highest eight address lines are important too. A zero on one of these lines selects a particular half-row of five keys:

IN Reads keys (bit 0 to bit 4 inclusive)

#FEFE: SHIFT, Z, X, C, V
#FDFE: A, S, D, F, G
#FBFE: Q, W, E, R, T
#F7FE: 1, 2, 3, 4, 5

#EFFE: 0, 9, 8, 7, 6
#DFFE: P, O, I, U, Y
#BFFE: ENTER, L, K, J, H
#7FFE: SPACE, SYM SHIFT, M, N

A zero in one of the five lowest bits means that the corresponding key is being pressed. If more than one address line is made low, the result is the logical AND of all single inputs, so a zero in a bit means that at least one of the appropriate keys is pressed. For example, only if each of the five lowest bits of the result from reading from port 00FE (for instance by XOR A/IN A,(FE)) is one, no key is pressed.

A final remark about the keyboard. It is connected in a matrix-like fashion, with 8 rows of 5 columns, as is obvious from the above remarks. Any two keys pressed simultaneously can be uniquely decoded by reading from the IN ports, however, if more than two keys are pressed decoding may not be uniquely possible. For instance, if you press Caps shift, B and V, the Spectrum will think also the Space key is pressed, and react by giving the 'Break into Program' report. This matrix behaviour is also emulated - without it, Zynaps for instance won't pause when you press 5,6,7,8 and 0 simultaneously.

Bit 5 (value 64) of IN-port FE is the ear input bit. When the line is silent, its value is zero, except in the early Model 2 of the Spectrum, where it was one. When there is a signal, this bit toggles. The Spectrum loading software is not sensitive to the polarity of this bit (which it definitely should not be, not only because of this model difference, but also because you cannot be sure the tape recorder doesn't change the polarity of the signal recorded!) Some old programs rely on the fact that bit 5 is always one (for instance Spinads); for these programs the emulator can mimic a Model 2 Spectrum.

Bits 6 and 7 are always one.

The ULA with the lower 16K of RAM, and the processor with the upper 32K RAM and 16K ROM are working independently of each other. The data and address buses of the Z80 and the ULA are connected by small resistors; normally, these do effectively decouple the buses. However, if the Z80 wants to read of write the lower 16K, the ULA halts the processor if it is busy reading, and after it's finished it lets the processor access lower memory through the resistors. A very fast, cheap and neat design indeed!

If you run a program in the lower 16K of RAM, or read or write in that memory, the processor is halted sometimes. This part of memory is therefore somewhat slower than the upper 32K block. This is also the reason that you cannot write a sound- or save-routine in lower memory; the timing won't be exact, and the music will sound harsh. Also, INning from port FE will halt the processor, because the ULA has to supply the result. Therefore, INning from port FE is a tiny bit slower on average than INning from other ports; whilst normally an IN A,(nn) instruction would take 11 T states, it takes 12.15 T states on average if nn=FE. See below for more exact information.

If the processor reads from a non-existing IN port, for instance FF, the ULA won't stop, but nothing will put anything on the data bus. Therefore, you'll read a mixture of FF's (idle bus), and screen and ATTR data bytes (the latter being very scarce, by the way). This will only happen when the ULA is reading the screen memory, about 60% of the 1/50th second time slice in which a frame is generated. The other 40% the ULA is building the border or generating a vertical retrace. This behaviour is actually used in some program, for instance by Arkanoid, and the emulator also emulates this behaviour.

Finally, there is an interesting bug in the ULA which also has to do with this split bus. After each instruction fetch cycle of the processor, the processor puts the I-R register 'pair' (not the 8 bit internal Instruction Register, but the Interrupt and R registers) on the address bus. The lowest 7 bits, the R register, are used for memory refresh. However, the ULA gets confused if I is in the range 64-127, because it thinks the processor wants to read from lower 16K ram very, very often. The ULA can't cope with this read-frequency, and regularly misses a screen byte. Instead of the actual byte, the byte previously read is used to build up the video signal. The screen seems to be filled with 'snow'; however, the Spectrum won't crash, and program will continue to run normally. There's one program I know of that uses this to generate a nice effect: Vectron. (which has very nice music too by the way). This effect has not been implemented however - it's a bit useless (but maybe I'll include it in the future).

The processor has three interrupt modes, selected by the instructions IM 0, IM 1 and IM 2. In mode 1, the processor simply executes a RST #38 instruction if an interrupt is requested. This is the mode the Spectrum is normally in. The other mode that is commonly used is IM 2. If an interrupt is requested, the processor first builds a 16 bit address by combining the I register (as the high byte) with whatever the interrupting device places on the data bus. The word at this address is then called. Rodnay Zaks in his book 'Programming the Z80' states that only even bytes are allowed as low index byte, but that isn't true. The normal Spectrum contains no hardware to place a byte on the bus, and the bus will therefore always read FF (because the ULA also doesn't read the screen if it generates an interrupt), so the resulting index address is 256*I+0FF. However, some not-so-neat hardware devices put things on the data bus when they shouldn't, so later programs didn't assume the low index byte was 0FF. These programs contain a 257 byte table of equal bytes starting at 256*I, and the interrupt routine is placed at an address that is a multiple of 257. A useful but not so much used trick is to make the table contain FF's (or use the ROM for this) and put a byte 18 hex, the opcode for JR, at FFFF. The first byte of the ROM is a DI, F3 hex, so the JR will jump to FFF4, where a long JP to the actual interrupt routine is put.

In interrupt mode 0, the processor executes the instruction that the interrupting device places on the data bus. On a standard Spectrum this will be the byte FF, coincidentally (...) the opcode for RST #38. But for the same reasons as above, this is not really reliable.

The 50 Hz interrupt is synchronized with the video signal generation by the ULA; both the interrupt and the video signal are generated by it. Many programs use the interrupt to synchronize with the frame cycle. Some use it to generate fantastic effects, such as full-screen characters, full-screen horizon (Aquaplane) or pixel colour (Uridium for instance). Very many modern programs use the fact that the screen is 'written' (or 'fired') to the CRT in a finite time to do as much time-consuming screen calculations as possible without causing character flickering: although the ULA has started displaying the screen for this frame already, the electron beam will for a moment not 'pass' this-or-that part of the screen so it's safe to change something there. So the exact time in the 1/50 second time-slice at which the screen is updated is very important. Because the emulator updates the screen at once, no single best solution can be given, and therefore the user can select one of three possibilities (low, normal or high video synchronisation, corresponding to a screen update after 1/200, 2/200 or 3/200 of a (relative) second after a Z80 interrupt) which gives the best results. Try for instance Zynaps; with normal video synchronisation the top four or five lines of the background move out-of-phase with the rest, and your space-ship flickers in that region. With low video synchronisation the background moves smoothly but the sprites flicker in all parts of the screen. Only with high video sync everything moves smoothly and doesn't flicker.

This emulator does not try to emulate the really time-critical border pattern effects (except when loading, but the width of the loading stripes are not quite right because also PC video timings come into play), but maybe I'll include it in the future. I will need some hard data on video timings then, and I've figured these out recently. Here they are.

Each line takes exactly 224 T states. After an interrupt occurs, 64 line times pass before the byte 16384 is displayed. At least the last 48 of these are actual border-lines. I could not determine whether my monitor didn't display the others or whether it was in vertical retrace, but luckily that's not really important. Then the 192 screen+border lines are displayed, followed by about 56 border lines again. 56.5 border lines would make up exactly 70000 T states, 1/50th of 3500000. However, I noticed that the frequency of the 50 Hz interrupt (measured in 1/T states!) changes very slightly when my Spectrum gets hot (I think it has something to do with the relative change of the frequencies of the two crystals in the Spectrum), so the time between interrupts will probably not be exactly 70000 T states. Anyway, whether the final border block is of fixed or variable length doesn't concern us either, the timings of the start and end of the screen, which are the timings of real interest, are fixed.

Now for the timings of each line itself. I define a screen line to start with 256 screen pixels, then border, then horizontal retrace, and then border again. All this takes 224 T states. Every half T state a pixel is written to the CRT, so if the ULA is reading bytes it does so each 4 T states (and then it reads two: a screen and an ATTR byte). The border is 48 pixels wide at each side. A video screen line is therefore timed as follows: 128 T states of screen, 24 T states of right border, 48 T states of horizontal retrace and 24 T states of left border.

When an interrupt occurs, the running instruction has to be completed first. So the start of the interrupt is fixed relative to the start of the frame up to the length of the last instruction in T states. If the processor was executing a HALT (which, according to the Z80 books I read, is effectively many NOPs), the interrupt routine starts at most 3 T states away from the start of the frame. Of course the processor also needs some T states to store the program counter on the stack, read the interrupt vector and jump to the routine, but since I cannot determine that by only using the Spectrum, it is useless information by that very reason alone!

Now when to OUT to the border to change it at the place you want? First of all, you cannot change the border within a 'byte', an 8-pixel chunk. If we forget about the screen for a moment, if you OUT to port FE after 14326 to 14329 T states (including the OUT) from the start of the IM 2 interrupt routine, the border will change at exactly the position of byte 16384 of the screen. The other positions can be computed by remembering that 8 pixels take 4 T states, and a line takes 224 T states. You would think that OUTing after 14322 to 14325 T states, the border would change at 8 pixels left of the upper left corner of the screen. This is right for 14322, 14323 and 14324 T states, but if you wait 14325 T states the ULA happens to be reading byte 16384 (or 22528, or both) and will halt the processor for a while, thereby making you miss the 8 pixels. This exception happens again after 224 T states, and again after 448, an so forth. These 192 exceptions left of the actual screen rectangle are the only ones; similar things don't happen at the right edge because the ULA don't need to read things there - it has just finished!

As noted above, reading or writing in low ram (or OUTing to the ULA) causes the ULA to halt the processor. When and how much? The processor is halted each time you want to access the ULA or low memory and the ULA is busy reading. Of the 312.5 'lines' the ULA generates, only 192 contain actual screen pixels, and the ULA will only read bytes during 128 of the 224 T states of each screen line. But if it does, the processor is halted for exactly 4 T states.

5.2 The Interface I

The Interface I is quite complicated. It uses three different I/O ports, and contains logic to page and unpage an 8K ROM if new commands are used. I won't be very detailed here; you could refer to the source code of the emulator if you want to know some details, or read the 'Spectrum Shadow ROM Disassembly' by Gianlura Carri, published by Melbourne House - but don't expect the same level of detail as of Ian Logan and Frank O'Hara in their Rom disassembly book.

The ROM is paged if the processor executes the instruction at ROM address 0008 or 1708 hexadecimal, the error and close# routines. It is inactivated when the Z80 executes the RET at address 0700.

I/O Port E7 is used to send or receive data to and from the microdrive. Accessing this port will halt the Z80 until the Interface I has collected 8 bits from the microdrive head; therefore, it the microdrive motor isn't running, or there is no formatted cartridge in the microdrive, the Spectrum hangs. This is the famous 'IN 0 crash'.
Port EF is used for several things:

Bit	7	6	5	4	3	2	1	0
READ				busy	dtr	gap	sync	write prot.
WRITE			wait	cts	erase	$r/\overline{w}$	comms clk	comms data

Bits DTR and CTS are used by the RS232 interface. The WAIT bit is used by the Network to synchronise, GAP, SYNC, WR_PROT, ERASE, R/_W, COMMS CLK and COMMS DATA are used by the microdrive system. If the microdrive is not being used, the COMMS DATA output selects the function of bit 0 of out-port F7:

Bit	7	6	5	4	3	2	1	0
READ	txdata							net input
WRITE								net output rxdata

TXDATA and RXDATA are the input and output of the RS232 port. COMMS DATA determines whether bit 0 of F7 is output for the RS232 or the network.

5.3 The SamRam

The SamRam contains a 32K static CMOS Ram chip, and some I/O logic for port 31. If this port is read, it returns the position of the joystick, as a normal Kempston joystickinterface would. If written to, the port controls a programmable latch chip (the 74LS259) which contains 8 latches:

Bit	7	6	5	4	3	2	1	0
WRITE					address			bit

The address selects on of the eight latches; bit 0 is the new state of the latch. The 16 different possibilities are collected in the diagram below:

OUT 31,	OUTPUT	RESULT
0	0	Switch on write protect of CMOS RAM
1	0	Write to CMOS RAM allowed
2	1	Turn on CMOS RAM (see also 6/7)
3	1	Turn off CMOS RAM (standard Spec. ROM)
4	2	—
5	2	Ignore all OUT's to 31 hereafter
6	3	Select CMOS bank 0 (Basic ROM)
7	3	Select CMOS bank 1 (Monitor,...)
8	4	Select interface 1
9	4	Turn off IF 1 (IF1 ROM won't be paged)
10	5	Select 32K RAM bank 0 (32768-65535)
11	5	Select 32K RAM bank 1 (32768-65535)
12	6	Turn off beeper
13	6	Turn on beeper
14	7	—
15	7	—

At reset, all latches are 0. If an `OUT 31,5` is issued, only a reset will give you control over the latches again. The write protect latch is not emulated; you're never able to write the emulated CMOS ram in the emulator. Latch 4 will pull up the M1 output of the Z80. The Interface I won't page the ROM anymore then.

5.4 The Z80 microprocessor

The Z80 processor is quite straightforward, and contains to my knowledge no interesting bugs or quirks. However, it has some undocumented features. Some of these are quite useful, and some are not, but since many programs use the useful ones, and a few programs use the weird ones, I tried to figure them out and emulate them as best as I could. There is a Z80 emulator around, intended as a CP/M emulator, which halts the program if an undocumented opcode is encountered. I don't think this makes sense. ZiLOG doesn't dictate the law, the programs which use the processor's features do!

Most Z80 opcodes are one byte long, not counting a possible byte or word operand. The four opcodes CB, DD, ED and FD are shift opcodes: they change the meaning of the opcode following them.

There are 248 different CB opcodes. The block CB 30 to CB 37 is missing from the official list. These instructions, usually denoted by the mnemonic SLL, Shift Left Logical, shift left the operand and make bit 0 always one. Bounder and Enduro Racer use them. The SamRam monitor can disassemble these and uses the mnemonic SLL. These instructions are quite commonly used.

The DD and FD opcodes precede instructions using the IX and IY registers. If you look at the instructions carefully, you see how they work:

2A nn	LD HL,(nn)
DD 2A nn	LD IX,(nn)
7E	LD A,(HL)
DD 7E d	LD A,(IX+d)

A DD opcode simply changes the meaning of HL in the next instruction. If a memory byte is addressed indirectly via HL, as in the second example, a displacement byte is added. Otherwise the instruction simply acts on IX instead of HL. (A notational awkwardness, that will only bother assembler and disassembler writers: JP (HL) is not indirect; it should have been denoted by JP HL) If a DD opcode precedes an instruction that doesn't use the HL register pair at all, the instruction is executed as usual. However, if the instruction uses the H or L register, it will now use the high or low halves of the IX register! Example:

44	LD B,H
FD 44	LD B,IYh

These types of inofficial instructions are used by very many programs. By the way, many DD or FD opcodes after each other will effectively be NOPs, doing nothing except repeatedly setting the flag 'treat HL as IX' (or IY) and taking up 4 T states. (But try to let MONS disassemble such a block.)

I've never seen a program using inofficial ED instructions, and except for ED 6B nn, a long version of 2A nn, LD HL,(nn) I don't know any. I am pretty sure however that they exist, but I never took the trouble to test them all.

About the R register. This is not really an undocumented feature, although I have never seen any thorough description of it anywhere. The R register is a counter that is updated every instruction, where DD, FD, ED and CB are to be regarded as separate instructions. So shifted instruction will increase R by two. There's an interesting exception: doubly-shifted opcodes, the DDCB and FDCB ones, increase R by two too. LDI increases R by two, LDIR increases it by 2 times BC, as does LDDR etcetera. The sequence LD R,A / LD A,R increases A by two, except for the highest bit: this bit of the R register is never changed. This is because in the old days everyone used 16 Kbit chips. Inside the chip the bits where grouped in a 128x128 matrix, needing a 7 bit refresh cycle. Therefore ZiLOG decided to count only the lowest 7 bits. Anyway, if the R register emulation is switched on the R register will behave as is does on a real Spectrum; if it is off it will (except for the upper bit) act as a random generator.

You can easily check that the R register is really crucial to memory refresh. Assemble this program:

```
     ORG 32768
     DI
     LD B,0
L1   XOR A
     LD R,A
     DEC HL
     LD A,H
     OR L
     JR NZ,L1
     DJNZ L1
     EI
     RET
```

It will take about three minutes to run. Look at the upper 32K of memory, for instance the UDG graphics. It will have faded. Only the first few bytes of each 256 byte block will still contain zeros, because they were refreshed during the execution of the loop. The ULA took care of the refreshing of the lower 16K. (This example won't work on the emulator of course!)

Then there's one other dark corner of the Z80 which has its effect on programs like Sabre Wulf, Ghosts'n Goblins and Speedlock. The Mystery of the Undocumented Flags!

Bit 3 and 5 of the F register are not used. They can contain information, as you can readily figure out by using PUSH AF and POP AF. Furthermore, sometimes their values change. I found the following empirical rule:

> The values of bit 7, 5 and 3 follow the values of the corresponding bits of the last 8 bit result of an instruction that changed the usual flags.

For instance, after an ADD A,B those bits will be identical to the bits of the A register. (Bit 7 of F is the sign flag, and fits the rule exactly). An exception is the CP x instruction (x=register, (HL) or direct argument). In that case the bits are copied from the argument.

If the instruction is one that operates on a 16 bit word, the 8 bits of the rule are the highest 8 bits of the 16 bit result - that was to be expected since the S flag is extracted from bit 15.

Ghosts'n Goblins use the undocumented flag due to a programming error. The rhino in Sabre Wulf walks backward or keeps running in little circles in a corner, if the (in this case undocumented) behaviour of the sign flag in the BIT instruction isn't right. I quote:

```
AD86   DD CB 06 7E          BIT 7,(IX+6)
AD89   F2 8F AD             JP P,#AD8F
```

An amazing piece of code! Speedlock does so many weird things that all must be exactly right for it to run. Finally, the '128 rom uses the AF register to hold the return address of a subroutine for a while. To keep all programs happy and still have a fast emulator, I had to make a compromise. The undocumented flags are not always emulated right, but they are most of the time.

Finally, a remark about the interrupt flip flops IFF1 and IFF2. There seems to be a little confusion about these. These flip flops are simultaneously set or reset by the EI and DI instructions. IFF1 determines whether interrupts are allowed, but its value cannot be read. The value of IFF2 is copied to the P/V flag by LD A,I and LD A,R. When an NMI occurs, IFF1 is reset, thereby disallowing further (maskable) interrupts, but IFF2 is left unchanged. This enables the NMI service routine to check whether the interrupted program had enabled or disabled maskable interrupts. So, Spectrum snapshot software can only read IFF2, but most emulators will emulate both, and then the one that matters most is IFF1.

Now for the emulated Z80. I have added eight instructions, to speed up the RS232 input and output of the Interface I and several things of the SamRam. These opcodes, ED F8 to ED FE are of little use to any other program. ED FF is a nice one: it returns you to DOS immediately. I used it for debugging purposes.

5.5 File formats

This sections describes the formats of the files used by the emulator.

ROMS.BIN:

00000-03fff	Ordinary Spectrum rom
04000-05fff	Interface I rom (8K)
06000-09fff	First SamRam rom (contains BASIC)
0a000-0dfff	Second SamRam rom (contains monitor,...)
0e000-11fff	First Spectrum 128K rom (active at RESET)
12000-15fff	Second Spectrum 128K rom (contains BASIC)

The ordinary rom has not been modified. The Interface I rom has undergone some modifications, to speed up the RS232 input/output routines. If you don't like this, or want to use another version of the Interface I, you could put that code at the right place in the ROMS.BIN file. The interface I should work properly, although the RS232 will be slower (always FORMAT the "b" or "t" channel at 19200 baud, by the way, if you replace the rom code, there's no point in waiting for nothing!) The microdrive routines have not been modified in any way. Here are the changes of the Interface I rom:

Address	Old	New	Address	Old	New
0B9E	ED	ED	0D20	FB	00
0B9F	5B	FC	0D2A	37	ED
0BA0	C3	F5	0D2B	F3	FD
0BA1	5C	C3	0D2C	CE	18
0BA2	21	34	0D2D	00	10
0BA3	20	0C	0D4C	FB	00

These changes are not likely to cause problems; there are several versions of the Interface I rom around, and program developers know this. It is also a bit pointless to check whether the Interface I rom hasn't been modified; who would put his snapshot software in there anyway, and that's what those people are afraid of.

The first and second SamRam rom have been modified more extensively. The biggest problem was that switching the upper 32K ram bank is very fast in reality, but on the PC two blocks of 32K bytes had to be REP MOVSWded. But since no programs know of the SamRam code anyway, this won't cause any more problems it wouldn't already cause either. The two Spectrum 128 roms have not been modified.

.TAP FILES:

The .TAP files contain blocks of tape-saved data. All blocks start with two bytes specifying how many bytes will follow (not counting the two length bytes). Then raw tape data follows, including the flag and checksum bytes. The checksum is the bitwise XOR of all bytes including the flag byte. For example, when you execute the line `SAVE "ROM" CODE 0,2` this will result:

$$\underbrace{\text{13 00}}_{1}\ \overbrace{\underbrace{\text{00}}_{2}\ \underbrace{\text{03}}_{3}\ \underbrace{\text{52 4F 4D 7x20}}_{4}\ \underbrace{\text{02 00 00 00 00 80}}_{5}\ \underbrace{\text{F1}}_{6}}^{\text{Spectrum-generated data}}\ \underbrace{\text{04 00}}_{7}\ \overbrace{\underbrace{\text{FF}}_{8}\ \underbrace{\text{F3 AF}}_{9}\ \underbrace{\text{A3}}_{10}}$$

with the following meaning:

1: First block is 19 bytes (17 bytes+flag+checksum)

2: flag byte (A reg, 00 for headers, FF for datablocks)

3: first byte of header, indicating a code block

4: filename

5: header info

6: checksum of header

7: length of second block

8: flag byte

9: first two bytes of rom

10: checksum ('checkbittoggle' would be better)

The emulator will always start reading bytes at the beginning of a block. If less bytes are loaded than are available, the other bytes are skipped, and the last byte loaded is used as checksum. If more bytes are asked for than exist in the block, the loading routine will terminate with the usual tape-loading-error flags set, leaving the error handling to the calling Z80 program.

Note that it is possible to join .TAP files by simply stringing them together, for example:

```
COPY /B FILE1.TAP + FILE2.TAP ALL.TAP
```

For completeness, I'll include the structure of a tape header. A header always consists of 17 bytes:

Byte	Length	Description
0	1	Type (0,1,2 or 3)
1	10	Filename (padded with blanks)
11	2	Length of data block
13	2	Parameter 1
15	2	Parameter 2

The type is 0,1,2 or 3 for a Program, Number array, Character array or Code file. A screen$ file is regarded as a Code file with start address 16384 and length 6912 decimal. If the file is a Program file, parameter 1 holds the autostart line number (or a number >=32768 if no LINE parameter was given) and parameter 2 holds the start of the variable area relative to the start of the program. If it's a Code file, parameter 1 holds the start of the code block when saved, and parameter 2 holds 32768. For data files finally, the byte at position 14 decimal holds the variable name.

.MDR FILES:

The emulator uses a cartridge file format identical to the 'Microdrive File' format of Carlo Delhez' Spectrum emulator Spectator for the QL. The following information is adapted from Carlo's documentation. It can also be found in the 'Spectrum Microdrive Book', by Ian Logan (co-writer of the excellent 'Complete Spectrum ROM Disassembly').

A cartridge file contains 254 'sectors' of 543 bytes each, and a final byte flag which is non-zero is the cartridge is write protected, so the total length is 137923 bytes. On the cartridge tape, after a GAP of some time the Interface I writes 10 zeros and 2 FF bytes (the preamble), and then a fifteen byte header-block-with-checksum. After another GAP, it writes a preamble again, with a 15-byte record- descriptor-with-checksum (which has a structure very much like the header block), immediately followed by the data block of 512 bytes, and a final checksum of those 512 bytes. The preamble is used by the Interface I hardware to synchronise, and is not explicitly used by the software. The preamble is not saved to the microdrive file:

offset	length	name	contents
0	1	HDFLAG	Value 1, to indicate header block
1	1	HDNUMB	sector number (values 254 down to 1)
2	2		not used
4	10	HDNAME	microdrive cartridge name (blank padded)
14	1	HDCHK	header checksum (of first 14 bytes)
15	1	RECFLG	- bit 0: always 0 to indicate record block - bit 1: set for the EOF block - bit 2: reset for a PRINT file - bits 3-7: not used (value 0)
16	1	RECNUM	data block sequence number (value starts at 0)
17	2	RECLEN	data block length (<=512, LSB first)
19	10	RECNAM	filename (blank padded)
29	1	DESCHK	record descriptor checksum (of previous 14 bytes)
30	512		data block
542	1	DCHK	data block checksum (of all 512 bytes of data block, even when not all bytes are used)

repeated 254 times

(Actually, this information is 'transparent' to the emulator. All it does is store 2 times 254 blocks in the .MDR file as it is OUTed, alternatingly of length 15 and 528 bytes. The emulator does check checksums, see below; the other fields are dealt with by the emulated Interface I software.)

A used record block is either an EOF block (bit 1 of RECFLG is 1) or contains 512 bytes of data (RECLEN=512, i.e. bit 1 of MSB is 1). An empty record block has a zero in bit 1 of RECFLG and also RECLEN=0. An unusable block (as determined by the FORMAT command) is an EOF block with RECLEN=0.

The three checksums are calculated by adding all the bytes together modulo 255; this will never produce a checksum of 255. Possibly, this is the value that is read by the Interface I if there's no or bad data on the tape.

In normal operation, all first-fifteen-byte blocks of each header or record block will have the right checksum. If the checksum is not right, the block will be treated as a GAP. For instance, if you type OUT 239,0 on a normal Spectrum with interface I, the microdrive motor starts running and the cartridge will be erased completely in 7 seconds. CAT 1 will respond with 'microdrive not ready'. Try it on the emulator...

.SCR FILES:

.SCR files are memory dumps of the first 6912 bytes of the Spectrum memory. A coordinate (x,y), x between 0 and 255 and y between 0 and 192, (0,0) being the upper left corner of the screen, corresponds to the pixel address

16384+INT (x/8)+1792*INT (y/64)-2016*INT (y/8)+256*y

The lowest three bits of x determine which bit of this address corresponds to the pixel (x,y). This bit-map constitutes the larger part of the screen memory, 256*192/8=6144 bytes. The final 768 bytes are attribute bytes. The address of the attribute byte corresponding to pixel (x,y) is

22528+INT (x/8)+32*INT (y/8)

The lowest three bits of the attribute byte control the foreground color (the color of the pixel if the corresponding bit is set), bits 3-5 control the background color, bit 6 is the bright bit and bit 7 is the flash bit: if it is set, every 16/50th of a second the ULA effectively flips the foreground and background colours.

.Z80 FILES:

The old .Z80 snapshot format (for version 1.45 and below) looks like this:

Byte	Length	Description
0	1	A-register
1	1	F-register
2	2	BC-register pair (LSB, i.e. C, first)
4	2	HL-register pair
6	2	Program counter
8	2	Stack pointer
10	1	Interrupt register
11	1	Refresh register (Bit 7 is not significant!)
12	1	Bit 0 : Bit 7 of the R-registers
		Bit 1-3 : Border colour
		Bit 4 : 1=Basic SamRom switched in
		Bit 5 : 1=Block of data is compressed
		Bit 6-7 : No meaning
13	2	DE-register pair
15	2	BC'-register pair
17	2	DE'-register pair
19	2	HL'-register pair
21	1	A'-register
22	1	F'-register
23	2	IY-register
25	2	IX-register
27	1	Interrupt flipflop (0=DI, otherwise EI)
28	1	IFF2 (not particiulary important...)
29	1	Bit 0-1 : Interrupt mode (0, 1 oder 2)
		Bit 2 : 1 = Issue-2-Emulation
		Bit 3 : 1 = Double interrupt frequency
		Bit 4-5 : 1 = High video synchronisation
		: 3 = Low video synchronisation
		: 0,2 = Normal
		Bit 6-7 : 0 = Cursor/Protek/AGF Joyst ick
		: 1 = Kempston joystick
		: 2 = Sinclair 1 joystick
		: 3 = Sinclair 2 joystick

Because of compatibility, if byte 12 is 255, it has to be regarded as being 1. After this header block of 30 bytes the 48K bytes of Spectrum memory follows in a compressed format (if bit 5 of byte 12 is one). The compression method is very simple: it replaces repetitions of at least five equal bytes by a four-byte code ED ED xx yy, which stands for "byte yy repeated xx times". Only sequences of length at least 5 are coded. The exception is sequences consisting of ED's; if they are encountered, even two ED's are encoded into ED ED 02 ED. Finally, every byte directly following a single ED is not taken into a block, for example ED 6*00 is not encoded into ED ED ED 06 00 but into ED 00 ED ED 05 00. The block is terminated by an end marker, 00 ED ED 00.

That's the format of .Z80 files as used by versions up to 1.45. Since version 2.01 emulates the Spectrum 128 too, there was a need for a new format.

The first 30 bytes are almost the same as the old versions' header. Of the flag byte, bit 4 and 5 have got no meaning anymore, and the program counter (bytes 6 and 7) are zero to signal a version 2.01 .Z80 file. So loading a new style .Z80 file into an old emulator will cause an error or a reset at the most.
After the first 30 bytes, an additional header follows:

Byte	Length	Description
30	2	Length of additional header block (contains 23)
32	2	Program counter
34	1	Hardware mode: 0=Spectrum 48K, 1=0+interface I, 2=SamRam, 3=Spectrum 128K, 4=3+interface I.
35	1	If in SamRam mode, bitwise state of 74ls259. For example, bit 6=1 after an `OUT 31,13` (=2*6+1) If in 128 mode, contains last OUT to 7ffd
36	1	Contains 0FF if Interface I rom paged
37	1	Bit 0: 1 if R register emulation on Bit 1: 1 if LDIR emulation on
38	1	Last OUT to fffd (soundchip register number)
39	16	Contents of the sound chip registers

Hereafter a number of memory blocks follow, each containing the compressed data of a 16K block. The compression is according to the old scheme, except for the end-marker, which is now absent. The structure of a memory block is:

Byte	Length	Description
0	2	Length of data (without this 3-byte header)
2	1	Page number of block
3	[0]	Compressed data

The pages are numbered, depending on the hardware mode, in the following way:

Page	In '48 mode	In '128 mode	In SamRam mode
0	48K rom	rom (basic)	48K rom
1	Interf. I rom	Interf. I rom	Interf. I rom
2	-	rom (reset)	samram rom (basic)
3	-	page 0	samram rom (monitor, ...)
4	8000-bfff	page 1	Normal 8000-bfff
5	c000-ffff	page 2	Normal c000-ffff
6	-	page 3	Shadow 8000-bfff
7	-	page 4	Shadow c000-ffff
8	4000-7fff	page 5	4000-7fff
9	-	page 6	-
10	-	page 7	-

In 48K mode, pages 4,5 and 8 are saved. In SamRam mode, pages 4 to 8 are saved. In 128 mode, all pages from 3 to 10 are saved. This version saves the pages in numerical order. There is no end marker.