RUI FERNANDO FERREIRA RIBEIRO

UWE June 1996

A Spectrum Emulator for MS-WINDOWS

UWE, B.SC. (HONS) COMPUTING FOR REAL-TIME SYSTEMS

A Spectrum emulator for MS-WINDOWS

© Rui Fernando Ferreira Ribeiro, June 1996, March 1997
University of West of England • Instituto Superior de Engenharia
rui@ipp.pt • i890478@groucho.dei.isep.ipp.pt
http://www.idt.ipp.pt/~rff-ribe

June 1996

Tutor • Craig Duffy
Project Supervisor • Nigel Gunton

ABSTRACT

The objective of the project described in this dissertation was to develop an emulation that would allow the flexible and transparent emulation of a Sinclair Spectrum 48K. This feat was achieved by modelling the target machine in the terms of a Z80 microprocessor machine, with all the Spectrum hardware.

This program should show itself extremely robust, compatible with the original machine and very easy to use.

ACKNOWLEDGMENTS

Designing computer emulation is always difficult, and trying to implement all its unknown aspects even a greater challenge. The author would like to acknowledge here:

- all the persons, who pointed a few mistakes and gave new ideas;
- Filipe Silva, who lent me all his Spectrum tapes, gave many suggestions trough the emulator early stages, used it from the very beginnings, and exchanged Spectrum snapshots with the author;
- Pedro Oliveira, for all the books, magazines and his sound card just when they were needed;
- Alexandre Ulisses, for the companionship and his PC hardware book. Without it, the utility to read Spectrum tapes would have never had seen the light of the day;
- Ian Collier for our discussions in the Spectrum newsgroup;
- Francisco Cotrina for it's special interest and suggestion for the emulator;
- Helena Leitao, for providing me with this excellent and unique opportunity in my life;
- Craig Duffy and Nigel Gunton for their continued support and advice;
- to all my friends and my family for putting up with me;
- Finally to my girlfriend, for everything we shared together.

A special acknowledgement is also due to Gerton Lunter for is superb Z80 emulator and great documents. At last but not least, to Sir Clive Sinclair for producing such an enjoyable machine.

All trademarks within this text are acknowledged.

This report was submitted to the Faculty of Mathematics and Computers, The University of West of England, as part of a final year project as an Erasmus student, at the course of B.Sc. (HONS) Computing for Real Time Systems.

PREFACE

This software has been designed as a complete Spectrum emulation. It can be used by a person who has never used a Spectrum before. For the users who had known it, this emulation will bring many memories of good old time's back.

This project is based on the author's experience and researches in his own Spectrum. It's perfectly possible to adapt this work for other Z80 based computers (most notably the Spectrum +2, which is almost the same thing).

The emulation has been tested carefully and is believed to be reliable. However inevitably, some errors may be found. The author will be grateful for any comments so that any future version may benefit from your experience. Any suggestions for improvements will be largely appreciated.

By now, WspecEm has been published in two magazines, and can be found in several places in the WEB. Just some comments, selected from the many received:

PC Zone, October 96, UK: WspecEm v1.12
"…a superb new little ZX Spectrum emulator …"

CD Zone, October 96, UK: WSpecEm v1.12
"Relive the glory days of Sir Clive's rubber-keyed behemoth with this eerily accurate emulator…"

PC User, February 97, Argentina: WspecEm v1.21
"WspecEm, un completissimo emulador de Spectrum para Windows."

Gremlin's Retrocomputing WEB page:, UK: WspecEm v1.21
"Wspecem – The coolest way to run a Spectrum on your PC."

Table of Contents

Chapter 1

1. Project overview

Why this project was chosen and a few details

The project chosen was to produce a Spectrum emulator running on Microsoft Windows. This should allow any 48k program to be run without modification. The project itself is heavily implementation oriented, although a lot of effort has gone into design in order to make it as efficient and reliable as possible.

Throughout this report Spectrum is used to mean the original Spectrum 48k and family of compatible computers and also Windows will mean Microsoft Windows, except where mentioned.

The author decided to do this project for several reasons:

. The author wanted to learn how to program for Windows and to make use of the consistent interface provided by Windows;

. It could be used by the author himself and others;

. the author always wanted to do one such emulator, and tried it earlier, but failed behind with lack of knowledge;

. this project could be accomplished in the time available;

. the project could be tested, corrected and improved with the huge number of spectrum programs available;

. there was a empty niche for the general public acceptance of the project, since there wasn't any Windows Spectrum emulator;

. at last, the author deep knowledge of the internals of this machine had also a great influence in the process of choosing this project.

Unfortunately, the author planned to write parts of it in assembly, but time constraints prevented from accomplishing this task. Nevertheless, before moving to England a small program was written in assembler to digitise Spectrum tapes.

1.1. Introduction to emulators

Increasingly, emulator production is being seen as a useful discipline in its own right. Supposing there are some programs proceeding from a specific alien machine. Possibly, it's no more in production, with hardware characteristics radically different from our current machine, and there is a need to find a way to run them on the current, more powerful computer.

To this effect, a program can be built for implementing the hardware characteristics of this machine, mapping the access to the underlying OS and hardware of the host machine.

The Z80 emulator has to implement basic services provided by Spectrum hardware. However, it can rely on some low-level support, such as disk handling from the host machine OS.

Emulating a computer goes beyond recognising the CPU's instructions and translating them to another CPU's arquitecture. An emulator needs to provide an environment in which specific hardware devices appear to be present.

The emulator should be able to run a program exactly as it would run on the native machine. Thus, it will effectively trick the user and the running alien binary into thinking they are dealing with the real machine. Basically, what it needs to do is to read one byte, determine to which instruction it belongs, carries on with its action, and request more bytes to process this interaction if necessary. At all times you will have a set of variables that map the register and

memory of the real processor, which should be affected exactly the same way as on the original one.

Normally, you write a dispatcher that reads bytes from the emulated CPU address space and branches to the appropriate opcode handler, in the case of WSpecEm, using arrays of function pointers. Memory reads are not trapped, only writes. All memory reads are done inline. Each opcode adjusts the cycle count, to allow us to know when it's time for an emulated interrupt.

The main purposes of this emulation are:

. to use all the Spectrum software, reading it directly from tapes if necessary;

. Providing the machine and Windows resources to our alien binary as he would see the Spectrum resources.

There are many Spectrum emulators available for multiple platforms (their number must be very near 30, now), but this one is the first ever to be developed for Windows. As suspected, there was a huge welcoming to an emulator to such platform, since Windows nowadays has one of the largest established user base.

This emulator fully emulates all know Spectrum aspects, including its most obscure ones, as Z80 strange features.

1.2. Introduction to the Spectrum

The original Spectrum was released on June 1982. It cost £175, and came with a TV modulator (you needed an external TV or monitor to use it), 48k of RAM, 16k of ROM, which included a Basic interpreter and a cassette interface. The Spectrum was a great machine and its amazing what it accomplished with a so simple hardware philosophy.

Date	Event
February 1980	ZX80 launched at £100
August 1980	ZX80 production ceased:100000 made, 60% sold overseas
March 1981	ZX81 launched at £69. Over one million sold in two years.
June 1982	ZX Spectrum Launched. £175 for the 48K model
January 1985	Spectrum Plus & Spectrum 128 launched

Table 1-1 Sinclair related Spectrum operation business, and precedents.

The architecture of the Spectrum is simple by today's standards. Its central component is a Z80 CPU, running at 4 MHz.

There were also some rather common additional peripherals. Sinclair, cursor and kempston joysticks are emulated.

1.3. Introduction to the Z80 CPU

The Z80 microprocessor was launched in 1976, by ZILOG. This processor has approximately 8000 transistors and 158 instructions, being 696 with the different addressing modes (officially, of course). Its most common format is DIL (Dual in Line) format, with 40 pins. It

needs only a 5V-power source to work. This processor has Non maskable Interrupt and three different modes of maskable interrupts. To keep the hardware design simple, it has inside the complete logic to refresh 7-bit dynamic RAMs.

The Z80 microprocessor was designed to be a replacement for the 8080, and to offer additional capabilities. Because of this design philosophy, the Z80 offers all the instructions of the 8080, plus additional instructions.

Its registers are not orthogonal. They are associated with some family of instructions or certain actions. In respect to the general-purpose registers, you can use them as a word, or as halves (i.e. in AF, A, in BC, B and C, in DE, D and E, in HL, H and L). You can also access IX and IY as halves, but this is not documented by ZILOG.

There exists also an alternate set of the general-purpose registers (AF', BC', DE' and HL'). It is interchangeable with the general-purpose registers. It cannot be accessed directly, to deal with its contents you must exchange it with the general-purpose register set.

Register	Common function
AF	Accumulator and flags
BC	Counter on loops Port addressing
DE	16 bit-arithmetic (aux.) Destination in memory transfers
HL	16 bit arithmetic Memory transfers Pointer

Table 1-2 General-purpose registers.

Register	Function
I	Interrupt vector
R	Memory refresh logic
IX	Index register
IY	Auxiliary Index register
SP	Stack pointer
PC	Program counter

Table 1-3 Index registers.

The Z80 had a reasonable good set of instructions. It included a novelty in its design: to allow for more potent set of instructions, four bytes (0CBh, 0DDh, 0EDh and 0FDh) were chosen as gateways to access the new expansions. This has a cost: you lose speed using these extra instructions, since some can be as long as four bytes. There is also a peculiarity of the instruction decoder, where it inserts the index after DDCB and FDCB and before the opcode instruction affected by this combination.

The Z80 has been widely used in the computer industry and it's still in production. Today it is produced by ZILOG, MOSTEK, SGS, NEC, and SHARP. It can be found almost anywhere. The author himself saw a Z80 compatible microprocessor measuring about 0.8 x 0.8-cm, in an arcade machine board.

There was a huge effort into emulating the Z80 to the maximum detail possible since this would allow running a vast amount of software. In fact, this software took a major role on refining and debugging the Z80 emulation engine and approximately 1500 programs can be found on the CD

present online at the ftp.idt.ipp.pt site, in the /pub/sinclair/snaps directory. They were all used to test this emulator.

Chapter 2

2. Specification

Project goals

The project was to produce a Spectrum emulation capable of running the majority, if not all the programs that could run on a standard Spectrum. To accomplish this, the emulator must fully implement its characteristics. Specifically the following must be emulated:

. CPU

The CPU is emulated through a set of primitives and functions. Its source it's mostly in the \SOURCES\Z80 directory, and it represents the bulk of this project. See further discussions of an interpretative emulator to understand how it works.

. Screen

The screen is emulated through a memory buffer that is painted on the screen at regular intervals. This buffer is filled with calls to writebyte(), whose address is translated to pixel co-ordinates. Its value corresponds to a colour value, which will point to a pre-defined palette filled with the corresponding colours on the host machine.

At the writebyte() function there is a kind of byte cache for the Spectrum bitmap video address area, where the video buffer is updated. New calculations are made only when the

address of the byte to be written changes. The cache is flushed when it's time to display the WinG buffer into the Windows screen. This design is such, to improve the emulator speed on programs that write to the video memory area pixel-per-pixel, instead of byte-per-byte. All the extra speed that we can gain will be welcome for the slowest programs.

The 8x8 pixel attribute block has to be emulated and as such a 8x8 block is redrawn with different colours when the equivalent attribute byte is written, but only if it differs from the current one. This block emulation also means the Spectrum hardware colour clash is present on a high-end PC.

The screen emulation is the second more complicated part of this project.

. Keyboard

The keyboard is implemented with an array that keeps the current state of the Spectrum keys. This state is in a format prepared to map its hardware position as seen by the ROM routines. This array is maintained by the main Windows cycle, where we get the messages of the keys pressed or released. When a keyboard port read happens, the corresponding data of this array will be returned, with some logical operations to recognise ghost key closures, as it would happen on the real machine.

. Joystick

Joystick emulation works with the PC keyboard or a real PC joystick. Some special keys correspond to movements or the fire button of the joystick. The Sinclair joystick worked feeding values on the keyboard ports that would correspond to keys, so it works in interaction with the keyboard emulation.

The kempston joystick emulation has a variable integer of his own, which is filled also on the Windows main cycle, and read when we ask to read a port that would activate this device. The format of this integer is such, that corresponds to the value the real Spectrum would return if we had indeed such movements of the joystick. This value is built from the corresponding key presses on the keyboard.

. Tape

The tape interface is emulated with a virtual tape file, in the format recognised by the original Spectrum load routine. A compatible C routine is called trough a callback opcode, which as soon as is finished will transfer control, back to the Z80 scheduler. When finished, it will leave the emulated Z80 registers and the memory, as the original routine would do at the original machine.

The virtual tape files are created with TAPE2TAP, an external utility program, also developed for this project in assembler, running under DOS. This program is able to sample data from the printer port with a special cable, which schematics are given with the project, or from a SoundBlaster Pro input port.

TAPE2TAP (the extra program handed with the project, see Appendix A for details) samples and interprets the received data, creating a file in the format expected by the tape ROM emulated routines.

Chapter 3

3. Design

Detailed design overview and technical considerations

There were some decisions which had to be made in the course of the project, since it's earliest inceptions. This chapter tries to discuss most of them and explain the decisions and their results.

3.1. Design overview

At this project, it was chosen to implement an interpretative emulator. An interpretative emulator consists of a lookup dispatch table and native code functions. The functions implement each (Z80) instruction, and entries in the dispatch table point to these functions.

The emulator operates by fetching an 8-bit Z80 instruction. (Instructions can be longer, but the first byte defines the instruction's function.) This value acts as an index to an entry in the dispatch table. If the first byte is a CB, FD or ED prefix, an alternate dispatch table will be used again. This works this way because these opcodes operate as gateways to different instruction sets in the Z80.

The function's native instructions complete the operation, and control returns to the emulator's main loop.

All this design does is interpret one Z80 instruction at a time, all the time, and is thus known as an interpretative emulator. Interpretative emulation is not efficient when sections of code are executed frequently (e.g. in tight loops). However, it provides the best compatibility with the Z80 software.

A complete binary code recompilation into the host machine code, which would produce the biggest performance boost cannot be used, since Spectrum programs make extensive use of self-modifying code, rendering this approach impracticable. [1]

The best approach would be to use a mix between the two methods, which is called dynamic recompilation. Dynamic recompilation offers better efficiency during emulation by recompiling sections of frequently used Z80 instructions into chunks of native code, except where it detects self-modifying code. Rather than interpret each Z80 instruction inside a loop, the emulator jumps to a native code block that performs the looping operation. Such techniques are beyond the scope of this project. They can provide a very significant boost into the performance of an emulator, especially when emulating complex microprocessors like the Motorola's 68000, since it's interpretative code would be very intricate. As a downside, dynamic recompilation implies that you have a greater control of the overall process, which can only be accomplished in a low-level computer programming

[1] Note: Warajevo Spectrum emulator for DOS claims to have a utility they call a compiler, but instead what it does is append the snapshot to a runtime version of the emulator.

language, assembler. If WSpecEm was written under assembler, it could be boosted about 2 times. With dynamic recompilation, this would possibly increase to about 6 times. [2]

3.2. Design goals

1. Execute virtually all Spectrum programs

For this purpose, the emulator implements virtually all the known Spectrum aspects.

2. Bug free

The program was developed with a modular design, to locate rapidly a function by its action. It also allows a new person to understand better the sources by itself, and of course maintaining the emulator.

Many programs have been tested, and when the author was convinced the emulator was on a relative stable state, it was released on the Internet. Several persons pointed a few problems, which causes were promptly found and corrected.

3. Easy of use for the final user

[2] To see such an emulator, look at the Executor Macintosh emulator. The timing methods were based in already existent emulators temploying these techniques.

Programs running under windows all share the same common interface, and the user works intuitively with it. Therefore, the design of the emulator interface followed Windows programs interface guidelines, to profit from this advantage.

4. Able to read different snapshot formats

There are already in existence several Spectrum emulators for different platforms. Unfortunately, many of them appear to be using its own private format (albeit lately the .SNA and .Z80 formats are becoming the de facto standard). One of the goals was to read almost all the formats used by them to have a large software base already available.

3.3. Global design

3.3.1. Program Block Structure

At the Sources directory can be found the necessary header files for the application and the C modules that need special access to the Windows functions. The Z80 low-level hardware emulation is also tied to these modules, because of performance matters. If it were designed such as to be independent from the hardware, it would be much slower. The Windows interface is implemented at wspecem.c, poke.c and speed.c.

File	Function
env.h	Z80 global environment
Iglobal.h	Functions prototypes
ivars.h	Instruction tables -- pointers to functions
quirks.h	definitions for portability of Z80 kernel
snap.h	definitions for loading and saving snapshots
wspecem.h	Windows definitions
z80.h	Registers and primitives
error.c	Dialog boxes for emulator errors
initmem.c	Initialise Spectrum memory
poke.c	Poke dialog interface implementation
sna_load.c	Snapshot and data transfer disk logic -- loading part
sna_save.c	Snapshot and data transfer disk logic -- saving part
speed.c	Speed dialog interface implementation
wspecem.c	Windows interface for WSpecEm emulator
video.c	Translates hardware screen addresses to the pixel coordinate system, fill screen buffer and implements Flash

Table 3-1 Files at sources directory.

The Z80 processor emulation and the high-level hardware emulation are at the sources\z80 directory. Each module implements a specific class of tasks or instructions, mostly with the help of the C macro definition capabilities. What was done, was to define a macro for each family of instructions, and implementing each function of this family, relying on the corresponding macro.

File	Function
bits.c	Z80 bit manipulation instructions
callret.c	call & return logic emulation
exctranf.c	exchange, block transfer and search block instructions emulation
flags.c	Z80 flags -- basic support routines
init.c	Initialise emulation
intr.c	Z80 interrupts -- basic support routines
inout.c	I/O instructions emulation
jump.c	Jump logic emulation
kernel.c	Z80 initialisation and main cycle - basic support routines
ld16bits.c	Z80 16 bit load instructions
ld8bits.c	Z80 8 bit load instructions
math16bi.c	Z80 16 bit arithmetic instructions
math8bit.c	Z80 8 bit arithmetic and logic instructions
mem.c	Z80 memory - basic support routines
misc.c	Z80 misc. arithmetic & CPU control instructions
nothing.c	Do nothing function to delay emulation
ports.c	Z80 I/O - low level routines
rotate.c	Z80 rotate and displacement instructions
stack.c	Z80 stack operations - basic support routines
shutdown.c	'shutdown' Z80 emulation

Table 3-2 Files at sources\z80 directory.

3.3.2. Global design issue

Some decisions had to be taken:

In the Z80 emulation part, each Z80 opcode was implemented as a single C function. If it was not the case and each function implemented a different family of instructions, the emulator would have to further analyse the bits of the opcode, and so the speed would be unbearable. Each instruction family is on a different C module, grouped by the criteria as it's displayed on the Mostek Z80 data sheets, for convenience of the developer.

A dispatcher reads bytes from the emulated CPU address space and branches to the appropriate opcode handler, in the case of WSpecEm, using arrays of function pointers. Read operations are not trapped, only writes. All memory reads are done inline. Each opcode adjusts the cycle count, to allow the emulation to know when it's time for an emulated interrupt.

When painting the screen at WSpecEm.c, WinG and DirectDraw both need an intermediate buffer to paint the windows client area, but anyway this is the better strategy.

At loading or saving data, an approach was used where you read it in chunks, which is effectively an implementation of a file caching scheme. This was made because load and save times were unbelievably slow, when they were performed only one byte at a time.

3.3.2.1.INTERRUPTS

As Windows is a multi-cooperative environment, opposed to the Z80 where we can only have one task running at the same time, there was need to find a way to receive interrupts.

What was decided was to create code that keeps track of the virtual T-states into each instruction.

When the number of virtual T-states equivalent to an interruption on the real machine is reached, we leave the Z80 main emulation loop and process all the Windows messages. The program counter and the stack will be also modified according to the current interruption mode (if interrupts are enabled) or interrupt type, and the main emulation loop will be performed again.

3.3.3. Accessing Spectrum programs

Currently, it's easy to come across Spectrum programs, since there are several WWW pages and ftp sites with virtually hundreds of snapshots and a couple of emulators for several platforms, including of course this emulator. Of all sites the most famous and the Spectrum official site is ftp.nvg.unit.no. The author also implemented a heavily used site, at ftp.idt.ipp.pt. There are also some commercial CDs with hundreds of snapshots on them. See the Spectrum FAQ, for an exhaustive list.

In case the user does not have Internet access, he can always try to read its own collection of tapes, with TAPE2TAP. The author used this utility to read several tapes into snapshot format. The author also shared this utility with the Warajevo authors (Serbia). They improved it [3] and included it with v1.45 of their emulator.

3.4. Detailed module design

The Z80 directory contains the sources to emulate a Z80 microprocessor that is the heart and the most intricate part of this project.

A few instructions will be described, since there are about 2000 of them emulated.

The emulator parses each Z80 instruction, decodes it to determine it's intended effect, and executes 386 instructions to produce the same result a ZILOG CPU should have produced.

Along the way, the emulators faithfully imitates the CPU registers, flags, internal arithmetic, and logic states. It was made a special effort, to optimise the Z80 main loop and keep it to the minimum size possible, to give us a reasonable fast emulation (see Z80/kernel.c execute()).

3.4.1. Processing the flags

During the operation of most instructions, the flags are affected to reflect the processor actions. The states of the operations should be the same as the real processor, in order for everything to work properly. For speed of access and updating, flags are stored as separate variables rather than different bits within one byte, only assembling register F from these variables when F itself is needed.

Some flags are easy to calculate, but with some instructions, others are not so straightforward. To improve the speed of the emulation a parity table with 256 entries was also created to determine the parity of a given byte.

---

[3] They made it more reliable, but they sacrificed flexiblity at handling tape protection systems.

Flag	Action
S (sign)	Set if the result is negative (most significant bit set)
Z (zero)	Set if the result is zero
X	Mirrors bit 6 of the result of last op
H (half carry)	Set if there is a carry/borrow between the low and high nibbles
Y	Mirrors bit 4 of the result of last op
PV (parity/overflow)	Set if the result has even parity (logical operations) Set if signed overflow has occurred (arithmetic operations)
N (add/subtract)	N (add/subtract)
C (carry)	Set if there is a carry/borrow out (arithmetic operations) Set according to the bit moved out (logical operations)

Table 3-3 the 8 bits in the Z80's flag register F, from b7 to b0.

Analysing the **AND A** instruction with a byte macro from Z80\math8bit.c:

```
#define and_r(r, TS) { \
  T(TS); \
   flags._H = 1; \
   flags._N = flags._C = 0; \
   flags._Z = !(A=A & (r)); \
   flags._P = parity(A); \
   flags._S = A & (UCHAR)BIT_7; \
   flags._X = A & (UCHAR)BIT_5; \
   flags._Y = A & (UCHAR)BIT_3; \
}
```

In the start of the function, the T states are updated according to the time of the instruction, as usual. As defined in the data sheets, H is 1, N and C will become 0, Z will be 1 if A is 0, and P indexes the parity table to get its value (at the Z80 we have even parity). Sign flag will get the value of the 7th bit, and the undocumented flags get the results of the 3rd and 5th

value of the last operation, as usual. The result of the binary operation and will become stored into A.

One more complicated, adding a register with a value.

```
#define add_a_r(r, TS) { \
   LOCAL UCHAR tmp; \
   \
   T(TS); \
   flags._H = ( ((A & (UCHAR)0xF) + ((r) & (UCHAR)0xF) ) > (UCHAR)0xF
); \
   flags._N = 0; \
   flags._C = ((tmp = (UCHAR)(A + (r)))<A); \
   flags._P = (~A^(r)) & (UCHAR)BIT_7 & (tmp ^  A); \
   flags._S = ((A = tmp) & (UCHAR)BIT_7); \
   flags._Z = !A; \
   flags._X = A & (UCHAR)BIT_5; \
   flags._Y = A & (UCHAR)BIT_3; \
}
```

Flag H will be set if there was a carry from the 3rd bit to the 4th. Flag N is defined as 0. Flag C will be 1 if there was a carry, that is in an addition the result cannot be a lower number than one of the operands. The others flags behave just like in the last operation. A will get the result of the addition.

3.4.2. Handling interrupts

In addition to decoding and executing instruction opcodes the CPU must also handle interrupts.

Event	IFF1	IFF2
RESET	0	0
DI	0	0
EI	1	1
LD A,I [P/V = IFF2]	same	same
LD A,R [P/V = IFF2]	same	same
NMI	0	same
RETN	IFF2	same
INT	0	0
RETI	same	same

Table 3-4 Interrupts 'switch' interrupts and copy.

The Z80 has three hardware interrupt modes, from which you can select by software:

- IM 0: (It is active by default when a reset is performed to the Z80 microprocessor)

In IM 0, the Z80 processor waits for an instruction opcode in the data bus (this mode was created for compatibility with the 8080). In the Spectrum 0xFF is inserted in the data bus, which is 0x38 opcode, so IM_0 = IM_1.

- IM 1:

In IM_1 the current, PC is pushed in the stack and interrupts are disabled, before jumping to address 038h.

- IM 2:

This mode works like IM 1, except that instead of address 038h, the PC will be fetched from the address formed with I (high part of address), and the contents of the data bus. The low part of the address is normally 0ffh in the Spectrum, and always has this value in this emulator. Some Spectrum peripherals sometimes feed other values, but a commonly used strategy was to make register I point to a block filled with the same value. For more details in this machine problem, please read the Spectrum Faq.

3.5. The Spectrum ROM

The Spectrum had a 16K ROM area, located on the first 16k. The corresponding memory area on this emulator is read-only. It is only accessed directly when we are reading the ROM from a file and patching it to with the callback opcode to intercept the LOAD routine.

WSpecEm has two levels of emulation: hardware emulation and ROM emulation. The only part of the ROM emulated is the tape load routine. This code follows closely the original routine. It is called by a callback opcode. It’s the only part of the ROM that is emulated with C programming.. As detailed at Appendix A, this routine will read virtual tape files, possibly created with the TAPE2TAP program.

At any other times, it’s the original ROM that is running bellow the Z80 emulator.

3.6. The display hardware

The module video.c emulates the Spectrum screen hardware. Specifically, this entails ensuring the Windows display is kept in sync with the display memory of the Spectrum and implementing its strange memory organisation.

The display memory has 6144 bytes, each bit corresponding to a screen pixel. This provides room for 256x192 pixels, or 49152 points. The attribute memory, which is following to the display area, has 768 bytes. This maps to a resolution of 24 lines per 32 columns. So effectively, there is a colour resolution area of an 8x8-pixel square.

Screen addresses: 16384-22527

Screen Address format: 010S SLLL RRRB BBBB

S - section (third)

L – character scan row

R - row

B - column

From this format, we can use:

COORDS (pixels)

y 11 111 111

S R L

x 11111 111

B bit number

Attribute address area: 22528-23295

Attribute address format:

L- line (0-23)

C - column (0-31)

ADDRESS FORMAT

0101 10LL LLLC CCCC

Data byte format: bit 7 - flash

bit 6 - bright

bits 5-3 paper

bits 2-0 ink

3.6.1. Updating the display

The most time consuming task is effectively transferring the contents of the display buffer to the windows screen, so a particular effort was made to reduce the number of times this has to happen.

Our main Windows cycle we will receive an image for Windows each time windows thinks the image must be updated. The emulator itself also sends a message to Windows. The Windows main cycle will later receive this message when it’s the time for another frame, and the variable WindowDirty is true (i.e. the Spectrum screen has been modified).

Albeit we have on the Spectrum a hardware frame rate of 50 images per second, in this emulator we have as default the maximum value of 12 images per second to save speed. It can also be configured to 50 images per second by the user. If the user owns a slower machine it can decrease even further the frame rate, but doing so, it will degrade severely the image quality.

3.6.2. ports.c .

In the Spectrum, each bus address bit is assigned to a peripheral. When a port is read or written to and this bit is low, this means the corresponding peripheral will be activated, if it supports the current operation. Joystick, per example, only is activated when reading ports.

Bit	Peripheral
A0	ULA
A1	unused
A2	ZX Printer
A3	Microdrives and IF1
A4	Microdrives and IF1
A5	Kempston joystick
A6	Unused
A7	Unused
A8-A15	Keyboard scan rows

Table 3-5 Address bus bits that activate peripherals.

3.6.2.1. THE ULA

The ULA is a Sinclair specific chip that allows the microprocessor to relate with the tape, keyboard, speaker and the screen TV hardware scan. It also sends an interrupt signal to the Z80 microprocessor every 20ms. From the point of view of the programmer and the emulator, it is activated when the address of port is even. This will happen if another peripheral does not take precedence over it, as is the case when you activate simultaneously the ULA and a kempston joystick. Its official address port is 0FEh.

When outputting, bit b4 is the sound and the lower three bits are the border colour. When inputting, the lower five bits are the keyboard half scan row code.

3.6.2.2. THE KEYBOARD

The keyboard can be read trough port 0feh. It is arranged into 8 scan rows, 2 per line, with 5 keys. In both of them, the key data is in the lower 5 bits returned. The keyboard addresses and associated data bits are depicted in the next table. To associate the next table with the real keys, please look at the next figure, the Spectrum keyboard. The corresponding data bus bits to the real keyboard are D0 D1 D2 D3 D4 for the left half and D4 D3 D2 D1 D0 for the right half scan row.

Left	Right
A11	A12
A10	A13
A9	A14
A8	A15

Figure 3-1 Bits to activate corresponding half scan row.

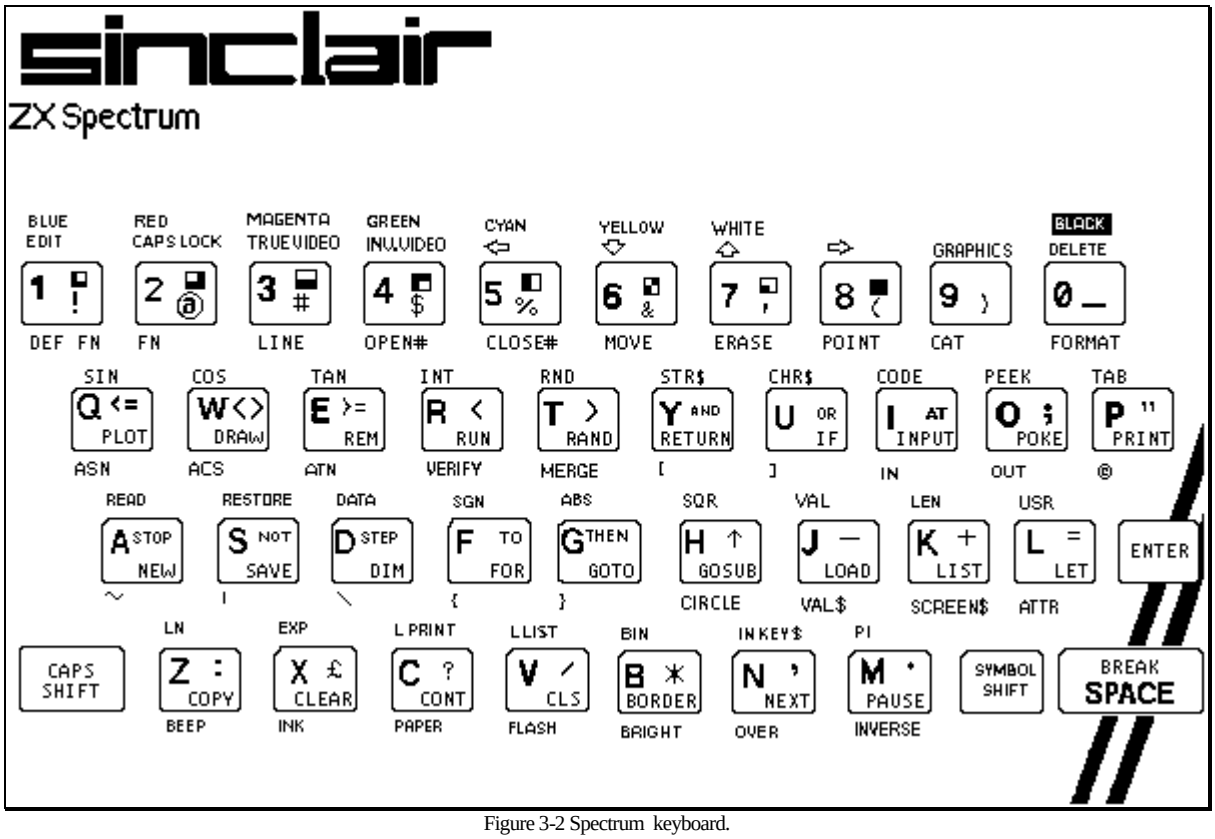

Figure 3-2 Spectrum keyboard.

The joystick port is activated which has in its address the bit 5 reset.

Data Bits	Meaning
b7	0
b6	0
b5	0
b4	fire
b3	up
b2	down
b1	right
b0	left

Table 3-6 kempston joystick binary data.

As was discovered while developing the emulation and already mentioned, if both the ULA and the kempston joystick are activated by a non-standard port, the joystick will take precedence over the ULA.

In respect to the other joysticks types, they map into certain spectrum keyboard keys, so they are no more than a specific case of the keyboard emulation. Note that in this case, the movement (e.g. the direction or button press) of a real joystick will simulate a Spectrum key. That's exactly how things happen in the real machine.

Chapter 4

4. Implementation notes

More technical details

This chapter talks about development strategies and how our emulated machine differs from the real one.

4.1. Differences between the emulator and the real machine

At the present, the only differences are:

1. Port 255 is not fully emulated, but is partially emulated. What this means it that some games can possibly run slower than in the real machine;
2. The hardware screen scanning is not emulated, so programs that depend on it to generate effects will have its output garbled. The more common effect is that in a few games some moving objects appear to be flickering, for short times.

4.2. Program testing and development

As always, a program has its own development cycle life, where it needs a special treatment.

The first goal was to implement all the Z80 official characteristics, and a mono Spectrum screen (e.g. the attribute area was not affecting the screen), in a DOS program, to not be affected by possible Windows errors.

To test this, the emulator was debugged until the ROM executed perfectly to the point of showing the normal boot screen, the Sinclair copyright. Next, it was implemented under Windows. After that the joystick was implemented and experiments started with a few ancient games, such as Pheenix, which practically only uses the most common instructions. This first phase enabled the emulation to reach a fair stable state with programs that do not use the Z80 undocumented instructions.

To further make experiments with the emulation, later the keyboard was implemented, and the 255 port.

More games were tried until all of them were working.

Finally, the colour was implemented (the attribute area) and all the undocumented instructions. This allowed running the most ill behaved games. When a game with problems was found, the emulator was debugged until the problematic instruction was found, and fixed.

To make the final test, it was released on the wild, and until now, people found a couple of bugs on it. The current version handed with the project appears to be very stable. It is being used across the world, and it has no known bugs, to the author knowledge.

Chapter 5

5. Conclusions

What was accomplished and the future of the project

In the opinion of the author the purposes of the project have been accomplished: the project has many users world-wide now, the author has received many appraisals, several suggestions, messages from people that are planning to improve it and even one message asking if the author was interested in a possible job in England. It was also published in two magazines (PC Zone and PC User). It WEB page at http://www.idt.ipp.pt/~rff-ribe/wspecem.html is always in use, specially when WspecEm is updated. As far as, it seems to be getting good feedback.

Overall, the project can be considered a success and it was implemented within the due date.

The author has learnt with this program not only how to program under Windows, but also more things about the Spectrum, the Z80, microprocessor theory and design, Internet, OOA and OOD methodologies. The author has searched for Spectrum related material, take active part on groups of discussion of emulators, and shared ideas with many people all over the globe. Without the Internet, this project wouldn't be in the state it is now.

It was released under the GNU licence agreement. Essentially it means people can improve it and modify it, as long they assume responsibility for their parts, and give credit to what was done by other people. Already people told they are willing to rewrite parts of it in

assembly, appending 128K, TC 2048 and TC 2068 emulation, and other several enhancements. Gilad Raz (gold@brm.co.il), already ported it for Win32, and implemented turbo edge loader detection.. The v2.30 32-bit port was somewhat loosely based in this contribution.

The TAPE2TAP utility was also improved by the Warajevo Spectrum emulator authors. It will be distributed with Warajevo v1.45.

Therefore, indeed, this will not be a project to die in a library shelf, but possibly the foundations for the best Spectrum emulator!

Glossary

Address

The address assigned through hardware of a specific memory location.

ALU

Arithmetic and logic unit. It performs arithmetic and logic instructions.

Assembler

Program that converts assembly language source code into machine code.

Assembly Language

A language composed by mnemonics understood by humans, that actually represents machine code instructions.

Bit

Binary digit.

Breakpoint

An instruction put in by a debugger that will return control to it when it is executed. Breakpoints can be done by patches (with native debuggers), hardware and/or emulation programs.

Bug

An undesired effect that the programmer and user did not intend to happen.

Carry flag

A flag used for arithmetic operations.

Clock speed

Every operation in a computer takes place to a heartbeat. A quartz crystal near the processor in your computer provides the beats. Each tick allows the computer to advance a step through the program it is running, a whole instruction (the Z80 does not have 1T state instructions, but the 386 does) or a subset of what is doing.

Crack

Writing a routine or executing some code that will enable the user to alter the behaviour or remove the protection of a program. In its simplest form, it will be only a patch.

Crash

A sequence of events that places the software into an unstable state. Normally halts or reboots the computer.

CPU

(Central Processing Unit) The CPU includes instruction decode logic, timing, control circuits, registers and the ALU.

Cross Assembler

An assembler program for a different processor of the machine running it.

Debug

The action of trying to get rid of a bug, or simply to get sure a program is correct.

Decrypter

A short program contained in a protection system that, when run will change apparent garbage into legitimate code. (See also self-modifying code.)

DirectDraw

A 32-bits graphics helper library tightly linked with the operating system (Windows 95 and NT). Provides a device independent way of getting a better graphic performance from Windows. It implements differential screen updates. (See also WinG)

DOS

Disk Operating System - a minimal operation system hacked by Microsoft. [It only provides a crude filesystem and basic memory management services].

It will be increasingly used only as a bootstrap code to load Windows and to provide backward compatibility with some old software and games. Starting in Windows 3.0 its functionality began moving to protected mode routines. Windows's 95 does not virtualises only a few DOS calls. Its API was inherited by Windows for better or worse. (See also Windows 3.11 and Windows'95)

Dynamic RAM

Dynamic memory uses a capacitor to keep its contents. Either it contains a charge or it does not. (See also refresh)

Emulator

If a program is called an emulator, it will behave and will be able to do most of the things the original did. In the specific case of a computer emulator, it will run its software producing the same results.

Fetch cycle

During the fetch cycle, the next byte is fetched from memory.

Garbage

A block of machine code that does not make sense. If the computer attempts to process it, it will almost certainly crash. (See also Crash, Decrypter.)

Hack

A special routine that is some kind of ability (something that was difficult to do or was not done in a proper way). Can possibly be used to mean something done in a very clever and unusual way. (See also crack.)

Headerless loader

A loader that does not contain the first "header part" of a file specifying it's name, length, … but has it ready built into memory. One of the first software protection methods devised in the Spectrum.

Interrupt

A small routine that is executed if the corresponding pin in the processor is activated by hardware and it is allowed (certain types of interrupts can be disabled by software). The clock interrupt is 1/50 of a second in the Spectrum and 1/28 of a second in the PC. (See also NMI)

Loader

Any program that reads a file [off tape, disk or whatsoever] into memory, and executes it.

Microcode

When a processor is dealing with a complicated instruction it breaks the job down into smaller, easier tasks. Some operations just pass through circuits but others need special software that is built into the processor. This is microcode.

NMI

Non-Maskable Interrupt. An interrupt that cannot be disabled by software.

Operating system

This was in the Spectrum the built-in program into the computer ROM to deal with all the basic things like reading the keyboard, loading software and provided a BASIC interpreter. Nowadays this definition has been extended and it must provide a whole bunch of services. (See also DOS, Windows 3.1)

Patch

The act of replacing a bit of code with other code that has a different meaning or behaviour. This patch may consist of a jump elsewhere in memory, a breakpoint, or a no operation opcode (NOP) to remove an unwanted operation (see also Breakpoint, Crack).

Program counter

Also known as PC. It points to the address of the next instruction to execute.

Refresh (cycle)

Dynamic RAM capacitors leak and the charge diminishes, so the refresh circuit has to monitor the RAM constantly, to recharge it. In the Z80, there is a built-in refresh circuitry to refresh 64K of dynamic RAM, and a special purpose register to drive it (the R register).

Self-modifying code

A program is a list of numbers held in memory, with each number having a special meaning. The processor can alter any memory location so there is no reason why those locations containing program code cannot be changed. This is self-modifying code and can be very efficient for writing tight loops. Instead of having to jump from routine to routine, a program that can alter itself will save many overheads; but the code becomes very difficult to debug.

Simulator

A simulator emulates the look and feels of, in our case, a computer, but probably is only a front end to the host operating system. One good example would be drivers to the Windows interface to make it look like or behave like the Macintosh.

Stack Pointer

The stack pointer (SP) is pointing permanently to a memory direction chosen by software. This position is the stack area where are automatically stored the return addresses of the call instructions, or the data manipulated with PUSH and POP routines. The stack area behaves like a LIFO.

Trace

Looking through a block of code while executing it in order to find some specific event or data.

T-State

Time state. A pulse of the CPU clock.

Windows 3.1

The first usable version of the GUI based operating system from Microsoft [running in top of DOS]. It was written following guidelines for software: independence from the [PC] hardware, virtual memory and task virtualisation. (Working closely with the Intel 386 and upper processors: not a coincidence because Microsoft helped in the 80386 design process.)

Windows 3.11

An improved version of Windows 3.1 with pre-beta code from the Chicago project [code name of the Windows'95 product]. Its main strength was the file-system related DOS calls virtualised in protected mode [read here File Access 32 Bits].

Windows' 95

The new version of the GUI based operating system from Microsoft. It is much more closely integrated with DOS. It is faster than previous versions, more robust, provides more memory for resources, adds several new utilities and enhances others. In rough lines it's Windows 4.x. For the future it will be even more stable and faster, now that is known that the Pentium brings new (undocumented) instructions to improve the security and performance of virtual machines.

Windows NT

A GUI based operating system from Microsoft. It is the most robust of all versions. In top of the Windows 95 characteristics, it provides true multitasking, device virtualisation, user authentication, and a separate virtual machine for each process. It is no longer has DOS running bellow it, but is able to run DOS in a virtual machine.

WinG

Graphics helper library for 16 and 32 bits. Provides a device independent way of getting a better graphic performance from Windows. It implements differential screen updates. It is freeware from Microsoft. (see also DirectDraw).

Bibliography

Baker (1985), *Dominio do Codigo Maquina (Mastering Machine Code On your ZX Spectrum)*
Editorial Verbo, Lisbon-Portugal.

Conger (1992), *Windows API Bible*
Waite Group Press

Duncan et al (1990), *Extending DOS*
Addison-Wesley Publishing Company, Inc.

Francisco (1984)*, El microprocesador Z80 (The Z80 Microprocessor)*
MicroHobby, Hobby Press, Barcelona-Spain.

Hutt (1994), *Object Analysis and design: description of methods*
Object Management Group, John Wiley & Sons, Inc.

Logan & O'Hara (1983), *The Complete Spectrum ROM disassembly*
Melbourne House (Publishers) Ltd., London.

Miller (1990), *Clive in his bike*,
Personal Computer World, vol.13 n.10, October (pp. 163-166).

Petzold (1992)*, Programming Windows Third Edition*
Microsoft Press, Washington.

Richter (1991), *Windows 3: A Developer's Guide*
M&T Books.

Schulman (1994), *Unauthorised Windows 95 Developer's Resource Kit*,
IDG Books: IDG Books Worldwide, Inc.

Schulman et al (1993), *Undocumented DOS Second Edition*
Addison-Wesley.

Simpson & Terrel (1983), *ZX Spectrum User's Handbook*
Butterworth & Co. (Publishers) Ltd., England.

Tischer (1992), *La Bible PC (PC Internals)*
Editions Micro Application, Paris-France.

Thompson (1995), *Building the Better Virtual CPU*
Byte, August 1995.

Thompson (1995), *Animation Techniques in Win32: a C++ programmer's guide to DIBs, palettes and sprites*.
Microsoft Press, Redmond, Washington

Zaks (1982), *Programming the Z80*,
Third Revised Edition, Sybex.

Further reading

Angell (1983), *Advanced graphics with the Sinclair ZX Spectrum*.
Macmillan

Baker (1983), *Mastering machine code on your ZX Spectrum*.
Interface

Bergin (1984), *Using graphics and sound on the Spectrum.*
Duckworth

Bert (1984*), Practical robotics and interfacing for the Spectrum.*
Granada

Bishop (1983), *Spectrum interfacing and projects.*
McGraw-H.

Carri (1985), *Spectrum Shadow ROM Disassembly*
Melbourne House (Publishers) Ltd.

Dickens (1983), *Spectrum Hardware Manual*
Melbourne House (Publishers) Ltd.

Grant (1984), *ZX programmer's guide*
Cambridge.

Hartnell (1983), *Instant Spectrum programming*
Interface

James (1984*), Expert guide to the Spectrum*
Granada

Kramer (1984), *The Spectrum Operating System*
MicroPress, Kent-UK.

Logan (1983), *Spectrum Microdrive Book*
Melbourne House (Publishers) Ltd.

Logan (1983), *Understanding your Spectrum: BASIC & machine code programming*
Melbourne

Scales (1984), *Spectrum peripherals guide.*
Pan

Scott (1984), *Complete Spectrum.*
Granada

Webb (1984), *Advanced Spectrum machine language*.
Melbourne

Appendix A

1. TAPE2TAP

This program was done as a preliminary project phase. It should work with all DOS versions starting from 3.0.

1.1. Theoretical introduction

In the Spectrum, the tape is relatively easy to use. The data is transmitted one bit at a time. The average medium speed (of the ROM routines) is 1610 bits per second. Theoretically, you could push it to 5000 bits per second with your own routines, as in most protection methods of the old days.

The leader pilot is normally repeated 256 times (about 2-5 seconds, 2168 T states on and off), followed by an only sync pulse (667 T states off and 735 T states on). Then the data pulse, logic zero (885 T states off and on) or logic one (1710 T states off and on) depending on the actual data being transmitted. From the speeds presented, we have a mean transmission time of about 1610Hz.

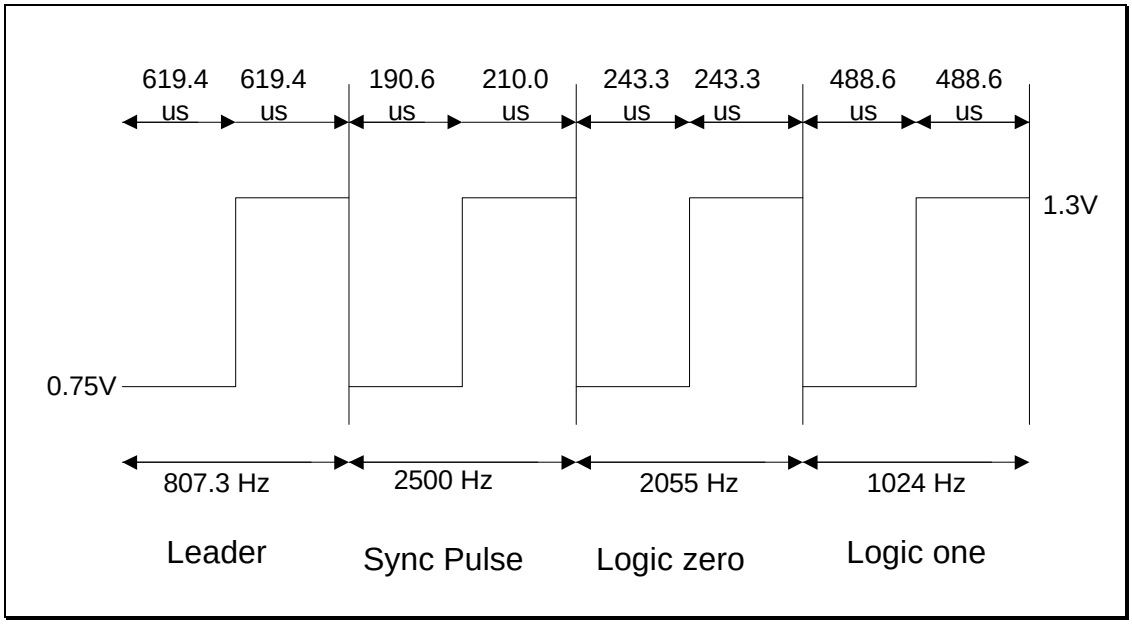

Figure A-1-1 Spectrum tape signal timings, measure at ULA pin 28

The first byte sent is a flag indicating a data/program or header block then the actual data, and finally the parity byte. The parity byte is built from all the other bytes that have been saved.

Tape2Tap works under DOS, independently of the processor speed, since it uses the Programmable Interrupt Timer to count the time, instead of depending on burning machine cycles for delays. The program will also auto detect a SoundBlaster and use it. If you do not have or do not want to use it, you can digitise data trough the printer port.

It controls the SoundBlaster directly, without any use of DMA (so in theory, it will work well with SoundBlaster compatibles).

The data is saved into the file W1.TAP at the current MS-DOS directory. If the file does not exist it will be created, else data will be appended to it. In case of loading error the user is notified, the faulty block is not written to disk, and the tape has to be rewind to the beginning of

that block. When Tape2Tap reads a tape header block successfully, it displays the familiar ‘Program:’ or ‘Bytes:’ message, followed by name stored in the block.

BYTE	NAME	DESCRIPTION
0	BLOCK TYPE	0 – BASIC 1 – NUMERIC ARRAY 2 - ALPHANUMERIC ARRAY 3 - C/M
1	Block name	
11	Block Length	
13	Auto run line number or initial address of data	. line number: if > 9999 it has no effect . if array 14th byte is letter
15	BASIC Program length	

Table A-1 Tape header block format.

The VGA hardware is manipulated at the low-level, to produce Spectrum look-like stripes, generated according to the input data.

While sampling the tape data, the program recognises the following keys:

Key	Action
ESC	exit program
S	turn PC speaker on
N	turn PC speaker off
1	disable CRC checking
2	enable CRC checking

Table A-2 Keys recognised by Tape2Tap while sampling data.

Care must be taken not to leave the program while a block is being loaded, because it will not become saved. The option 1 disables Spectrum like CRC checking, because one of the tape protection methods was to save a block with an incorrect CRC. Of course, it must be enabled again as soon as possible, to check for possible tape errors.

Appendix

B

Appendix B

1. WSpecEm OOA & OOD

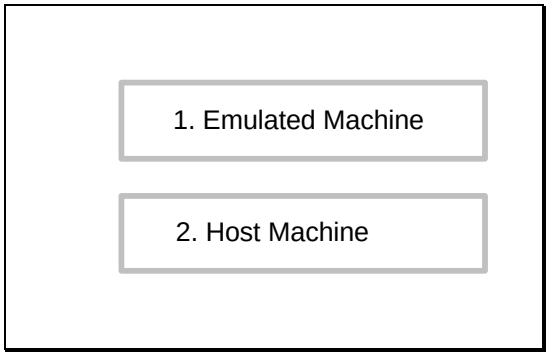

Figure B-1-1 WSpecEm - Subject layer

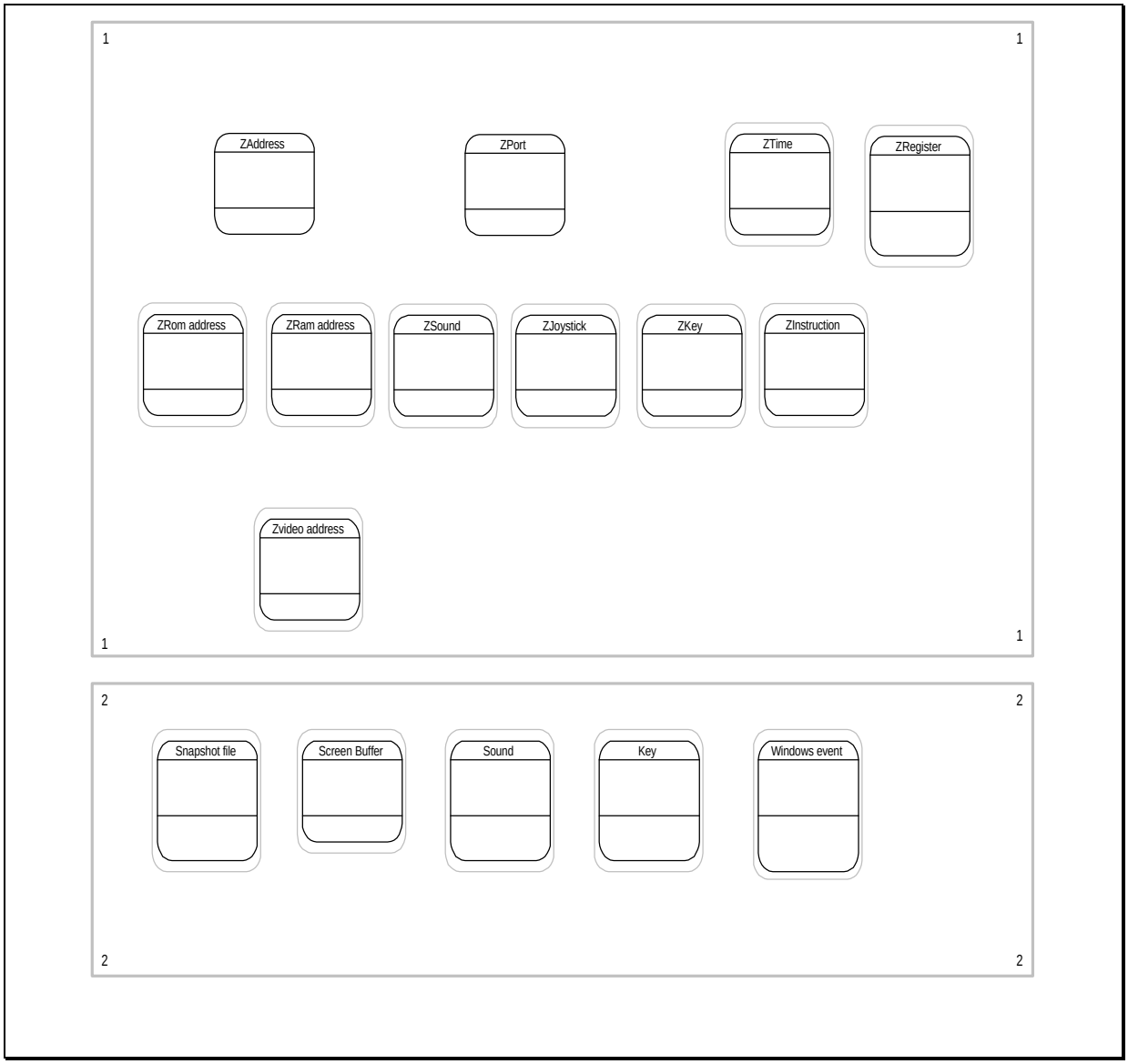

Figure B-1-2 WSpecEm - Subject and Class-&-Object layers

Class	Description
ZAdress	Z80 memory address space
ZROM address	Spectrum ROM address
ZRAM address	Spectrum RAM address
ZVideo address	Spectrum video memory address
ZPort	Z80 port address
ZSound	Spectrum port to output Sound
ZJoystick	Spectrum port address to read the Joystick
ZKey	Spectrum keys, 5 per each port
ZTime	Spectrum virtual time
ZRegister	Z80 processor registers
Zinstruction	Class that represents each Z80 opcode

Table B-3 Emulated Z80 class descriptions.

Class	Description
Snapshot file	In the emulation, we need to save and load data, in several formats.
Screen buffer	The host screen buffer, in our case the WinG-formatted buffer. Used to paint the screen.
Sound	The frequency of the sound, and actions to produce it.
Key	The host key.
Windows event	self-explicit.

Table B-4 Host machine class descriptions.

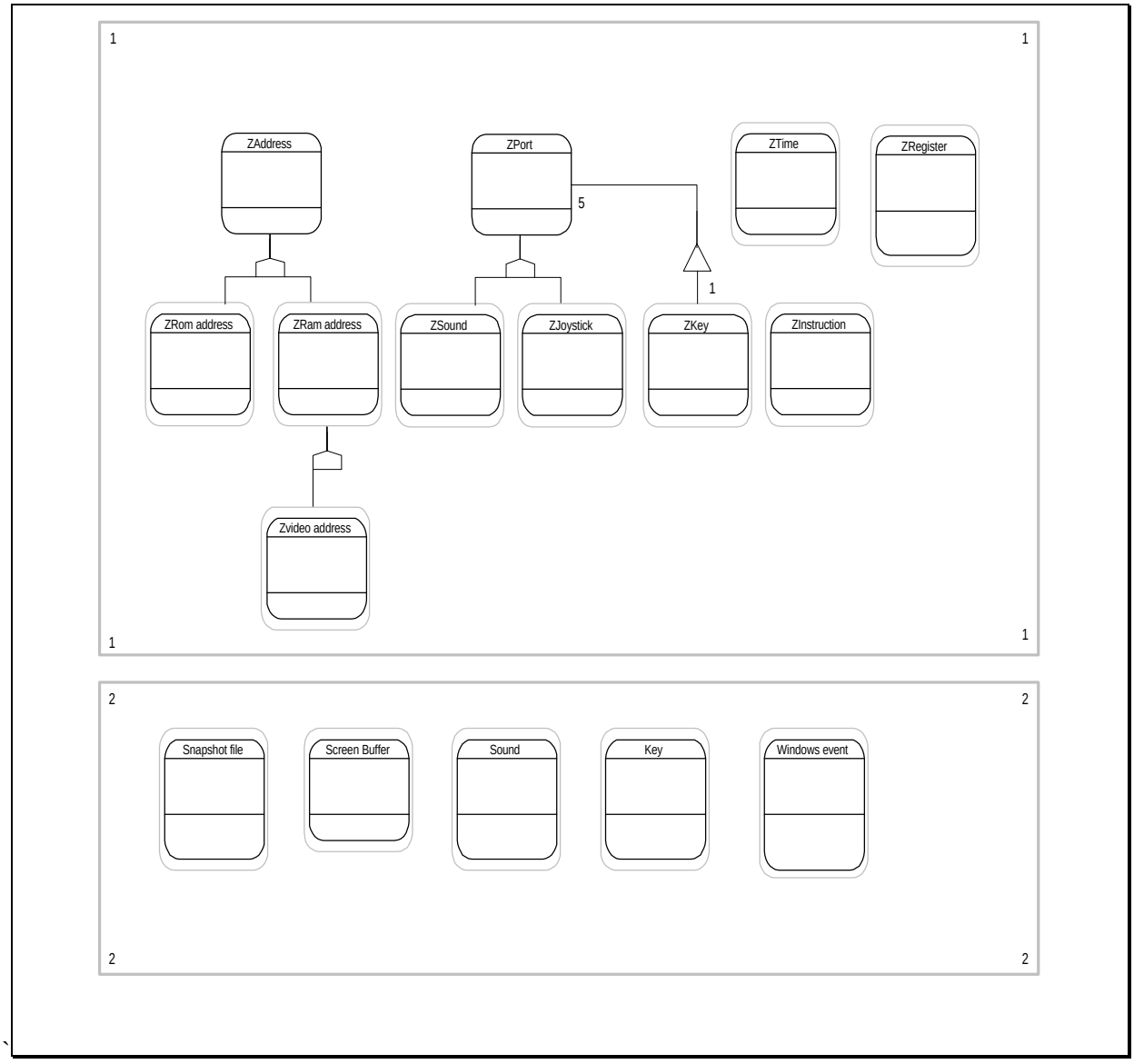

Figure B-1-3 WSpecEm - Subject, Class-&-Object, and Structure layers

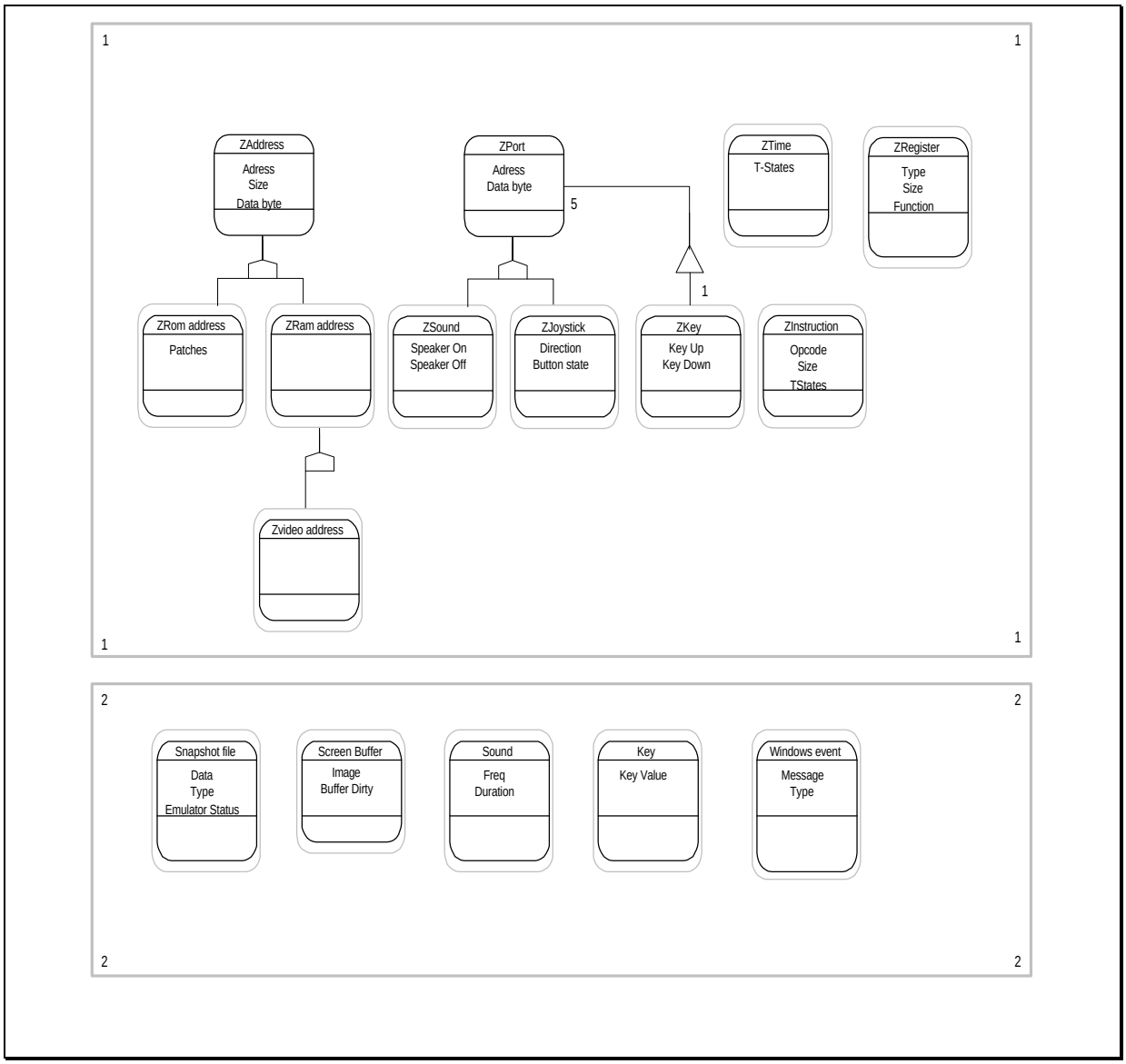

Figure B-1-4 WSpecEm - Subject, Class-&-Object, Structure, and Attribute layers

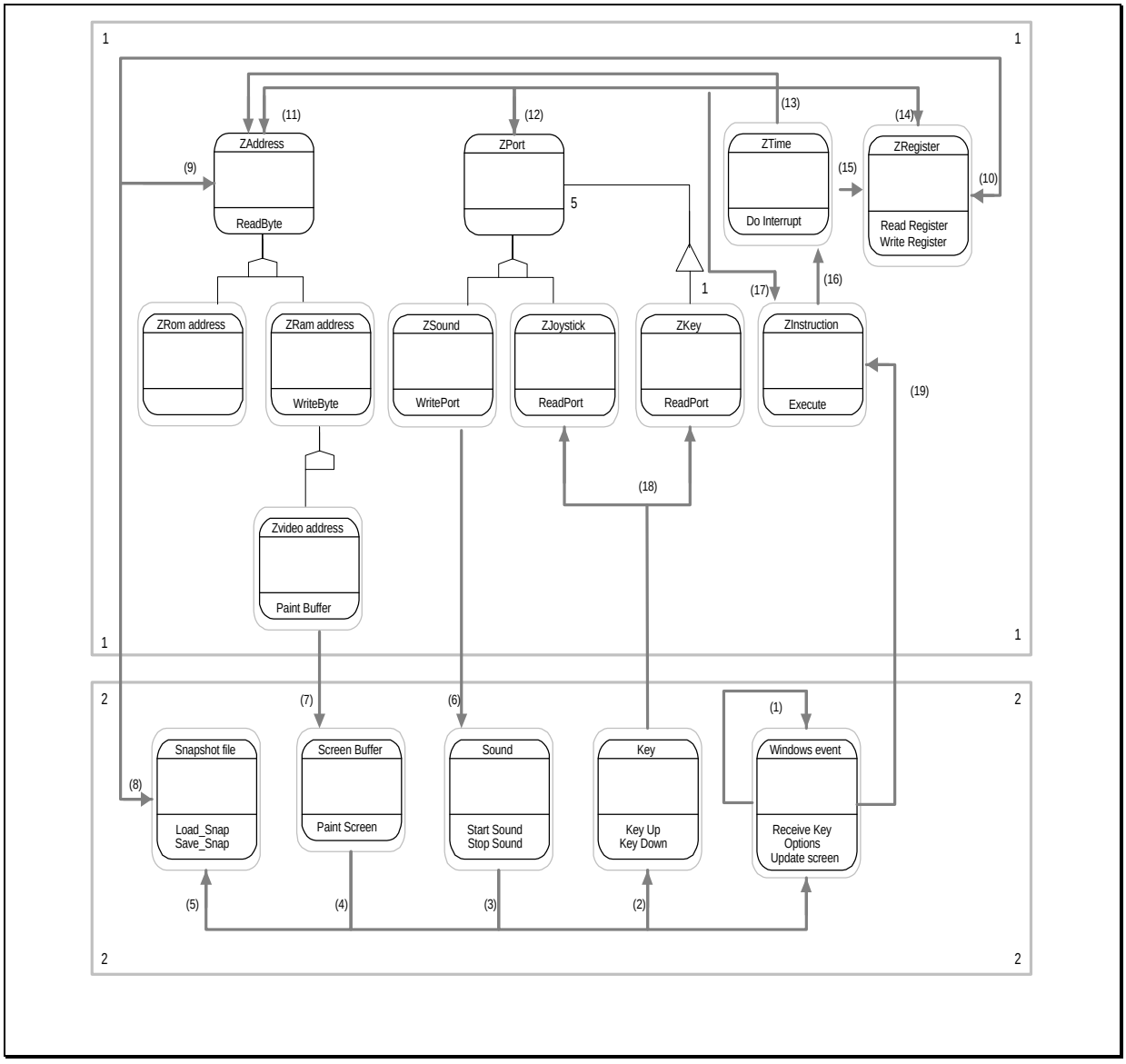

Figure B-1-5 WSpecEm - Subject, Class-&-Object, Structure, and Service layers.

1. WSpecEm windows code, in certain places send messages to the main Windows code itself, when it's answering certain events.
2. The main Windows events sends key codes to the class keys
3. The main Windows events gets orders to start or stop sound from the Sound class;
4. The main Windows class receives orders to paint the screen from itself and the class screen buffer;
5. The class snapshot files receives orders from the Windows event class to read or save snapshot files;

6. The sound class calculates it’s frequency from timing the bit outputs to the Zsound Spectrum port;
7. The image data is generated according to the contents of ZVideo address, which mimics closely the Spectrum native format video area;
8. When a snapshot file is to be saved, data is received from the Spectrum address area (Zaddress) and from the Z80 registers (ZRegister);
9. When a snapshot is being loaded, it’s data contents are transferred to the Z80 memory;
10. and the register are changed accordingly;
11. Z80 memory space can be affected by Z80 instructions;
12. Z80 Ports are affected by Z80 instructions;
13. The virtual Z80 time, keeps all T-states spent and when the time comes, it modifies the memory (the stack) to save the current PC;
14. The Z80 registers are the way to receive and send data to the Ports and the memory;
15. When it’s time for an interrupt, beside saving the current PC in memory, the Z80 registers are affected accordingly;
16. The time spent in the Z80 instructions is sent to ZTime, to track the Z80 virtual time;
17. The Z80 emulated instructions can effectively write or read ZAddress, ZPort, and ZRegister;
18. The host keys emulate the Spectrum native keyboard and the Spectrum joystick;
19. Windows events trigger the main Z80 execute cycle 50 times per second in emulated time.

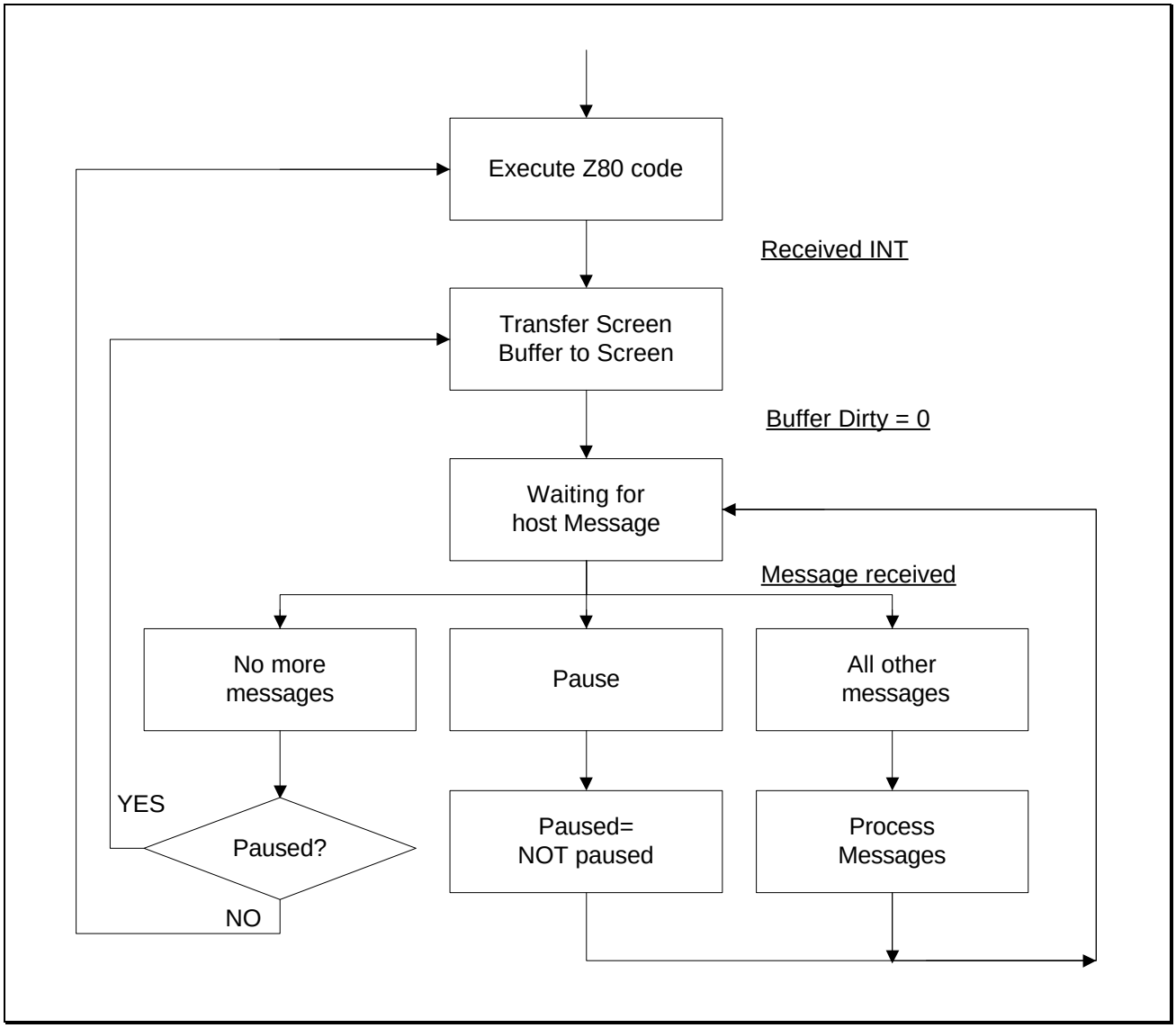

Figure B-1-6 Service chart for the emulator main cycle.

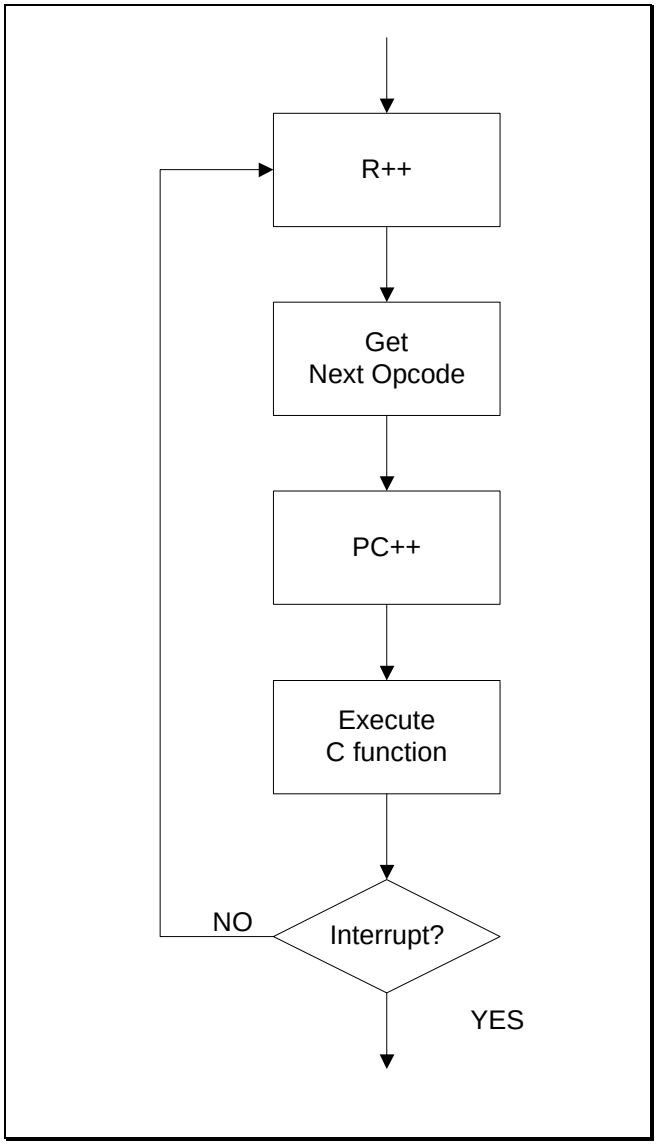

Figure B-1-7 Service chart for Execute task.

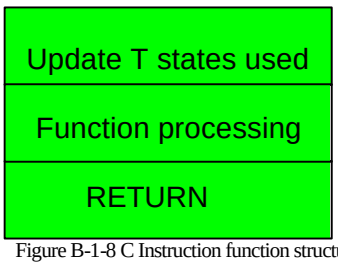

Figure B-1-8 C Instruction function structure in Z80 opcode emulation.

2. OOA, OOD and OOP

2.1. Introduction

The Yourdon OOA, OOD and OOP is a method that covers the fully objet-oriented development. The method does it with far less notation and documentation burden.

2.2. Purpose and scope

The method provides just the enough practical notations to get the job done and encourage creativity.

The method includes notations, activities and strategies.

OOA	domains classes and objects
OOD	human interaction, task and data management
OOP	strategies for implementing OOA and OOD results

Table B-1 OO services.

2.3. Principles

The method incorporates many principles for managing complexity, including: abstraction, encapsulation, generalisation-specialisation (and inheritance), association, communication with messages and scale. The three most significant principles are:

Methods of organisation	Organising things in an object oriented way
Categories of behaviour	Understanding and establishing what each object does
Personification	Brings together the last concepts into a whole

Table B-2 OO Methods.

The model is the multi-layer model. It consists of graphics and supporting text:

1. A graphical portion (often presented component by component, and by several layers combinations at a time)
2. Text (to specify what each object knows and does, including details of object interactions and scenarios).

2.4. Notation used

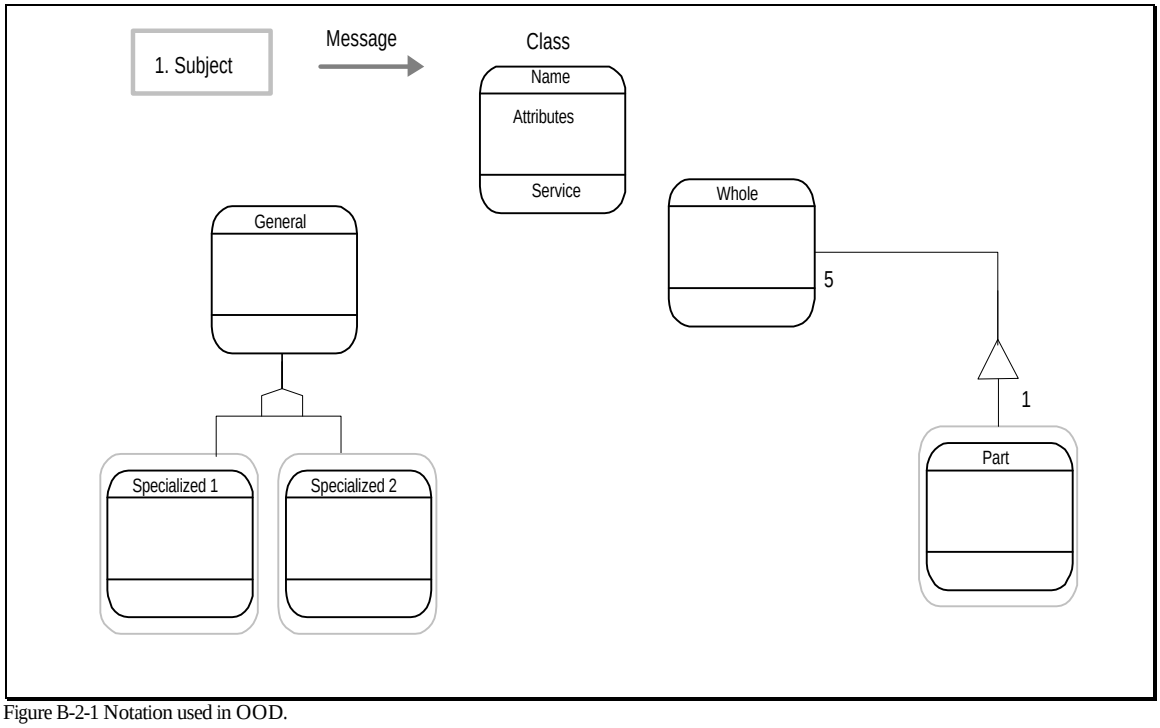

Figure B-2-1 Notation used in OOD.

This is the general OOD notation used it the design phase of WSpecEm. As an aside remark, when a class has a box around it, it means that this class has at least one instance. It's

just a coincidence that in this example the Whole and General classes do not have a box around it.

2.5. Multiple Layers

The multi-layer model consists of five layers. Layers are like acetate transparencies: one can look at one or some number of combinations of layers, one on top of another. Each layer may be toggled on or off, so that you can select the amount of complexity visible at one time.

Layer	Purpose
subject layer	identifies major grouping of classes.
class & object layer	shows the classes and their corresponding objects.
structure layer	shows generalisation-specialisation (inheritance) and whole-part.
attribute layer	shows what each object in a class knows: own attributes and connections with other objects. For real-time systems, the attributes are subdivided into state attributes, state-dependent attributes and state-independent attributes.
service layer	shows what each object in a class does: own services and messages it sends to get help from other objects. For real-time systems, the services are subdivided into state-dependent services and state independent services.

Table B-3 OOA layers purpose.

Appendix C

1. Initial project proposal

Project proposal

Purpose:

Emulate a Sinclair ZX-Spectrum 48K computer on a PC.

Why:

Windows is beginning to replace DOS as the environment of choice (mostly with the advent of Windows'95). Considering the degree of hardware independence it provides and that all Spectrum emulators are DOS programs, I want to be among the first ones writing one for Windows.

How:

I will develop the full Zilog's Z80 processor emulation code in Assembly (to compensate the loss of speed with the Windows maintenance tasks and interface) and the rest of the project in C.

Goals:

Mimic a ZX Spectrum 48K issue 2 with almost complete compatibility on any computer capable of running at least Windows 3.1 in enhanced mode;
Full emulation of Spectrum's screen, keyboard, Kempston joystick and beeper (trough Windows sound system device);
Z80 emulation will include all known unofficial instruction codes, flags and registers;
Built-in monitor/disassembler, mostly for debugging purposes (and for teaching).

Target:

Many former Spectrum users (myself included).

Minimum resources required:

A 486 DX/33 with 8Mb of RAM, a 200Mb hard disk (in the unlikely event of a standalone machine), equipped with a sound card;
Borland C++ v4.0;
Turbo Assembler v4.0;
Word for Windows v6.0.

Project developer:
Rui Fernando Ferreira Ribeiro (☎ 4804584)

2. Maintenance and Development notes

WSpecEm is a 48k Spectrum emulator that runs under Microsoft Windows. It emulates successfully a Model 2 or Model 3 Spectrum. It was developed over a 5-month period as my final year project, at University of West of England, Bristol, England.

The application was compiled using the MEDIUM memory model and the WinG package, under 16 bits. For 32 bits, it uses DirectDraw, so it does not any special files.

Firsts install the WING package and copy the WING.H file to the emulator source directory, if you want to compile it for 16 bits. Create a project, .ide or whatever your compiler calls it and includes there all c files you found in the directory of this file, Z80 directory and SND directory. Include also WSpecem.rc, Wspecem.def and Wing.LIB. The WinG.lib file that comes with WinG compiled straightforward under BC++ 4.52. If that is not your case, you can always create a Wing.LIB file with the IMPLIB program. Just run IMPLIB WING.DLL and it will create your WING.LIB file.

Note that the files extensions strings were not included as resources, because they are too long.

To correct or append more functions, always use the macros available in z80.h and env.h, in a way the program will stay easy to understand.

2.1. Features implemented

The following features have been implemented into WSpecEm:

. ‘sunken’ window border (v1.31)

. emulator fully 32-bit (Windows’95 and NT version);

. 32-bit version uses DirectDraw;

.in both versions, the screen update is now incremental (so the emulator works faster);

. capable of reading Z80’s v3.05 files;

. under custom installing, it allows to select the file name extensions that will be modified by the setup routine.

. Setup program for Windows'95;

. Windows help file;

. ZX's SNP snapshot file format;

. BMP file saving;

. real joystick support;

. keyboard template image with key press generator;

. Exact time delaying;

. program compiled for the 386 microprocessor.

. Automatic pause when the application is minimised to an icon.

. F4 as pause (v1.11);

. Emulator display frame rate definable (v1.10). Essential to run it on slower machines;

. Emulator has delay configurable by user (v1.10). Essential with new routines;

. ghost keys (a.k.a. ghost closures -- v1.03 -- they were documented but not introduced with v1.01 due to a lapse);

. Save the current Spectrum screen as a PCX file; (v1.01)

. Handles .BLK tape files (v1.01 - a bug prevented it from working before this version);

. all the Z80 undocumented instructions [that I'm aware of, at least];

. exact R register emulation;

. exact T state cycle timing in Z80 engine;

. kempston joystick as cursor keys with Fire as CTRL (if NUM LOCK off, the numeric keypad will work as a kempston joystick);

. Sinclair joystick as numeric keypad if NUM LOCK on (8 up, 2 down, 4 left, 6 right , 0 and 5 are both fire);

. recognises Windows'95 longfilenames;

. deals with ACH, PRG, SCR, SIT, SLT, SP, SNA, SNX, RAW, TAP, Z80, ZX and ROM spectrum related emulator files;

. ED FB level loader trap supported (a.k.a. DAT files);

. files may be loaded by drag-and-drop, double-clicking or with the File open menu;

. short-cut load and save keys, a' la Z80 (F2 - save and F3 - load, F5 - reset);

. Maps the Spectrum keyboard in the PC keyboard, with some extra PC-keys:

ESC : Caps-shif+SPACE (CTRL-BREAK in a Spectrum)
BACKSPACE : Same as Caps-shift+0 (DELETE in a Spectrum)
LEFT SHIFT : same as Caps-shift
RIGHT SHIFT : same as Symbol-shift
TAB : Same as Caps-shift + Symbol-shift (Cursor E)

+,-,*,/ extended keys, operate as their Spectrum Plus counterparts;

. Pause, reset, NMI functions;

. sound generation through PC speaker;

. Sound, flash, issue 3 or issue 2 control options;

. capability of saving options;

. WinG high-speed pixmap transfer;

. reissue to 1x1, 2x2, 3x3 or 4x4 size for optimum speed under WinG and DirectDraw;

. poke menu;

. easy of use;

. Enclosed DOS utility to sample Spectrum tapes in a PC as a .TAP file through a SoundBlaster or the printer port;

. Free, complete and commented sources available under GNU policy.

2.2. Features not implemented

The following features have not been implemented in WSpecEm:

. Border;

. High Colour Resolution (also known as cropping);

. Sound as WAVEFORM;

. Pasting on clipboard;

. does not save .TAP and .SLT files;

. trapping of the LOAD and SAVE routines to save blocks as files on disk 'a la Xtender;

. TAP editor;

. built-in debugger;

. interface I emulation;

. output to printer;

. 128k compatibility.

2.3. Third Party Copyrighted Material

ZX Spectrum 1982 ROM from AMSTRAD.
Thanks to AMSTRAD for granting permission to use the Spectrum ROM within software emulators.

WinG copyrighted Microsoft.

2.4. Tips

. If the application is not working, first make sure you have properly installed the WinG drivers. If you did, make sure you have the multimedia drivers installed (a sure fire way of doing that is by installing the Media Player Windows utility) -- remember that I am talking about video drivers,
. in windows'95 you can also remove or install the multimedia video drivers, so make sure you get them installed -- if you still want to check it out, find a little AVI file and see if you can play it;

. An 8-colour screen depth (256 colours) mode is recommended for optimum speed. If you want it faster and bigger, decrease the windows screen resolution;

. run it under the menu sizes 1x1, 2x2, 3x3 or you'll surely get very slow graphics transfer speed;

. to get a faster emulation and smoother sound quit all applications that you can, disable screen savers and remove any windows background bitmaps;

. to get a decent speed in the emulator, I recommend a Pentium or equivalent processor (although I suspect it working reasonably with a 50 MHz 486) --- currently with version 1.10 I think the 50MHz 486 is really true;

. and of course 8Mb or more for overall windows performance;

. if you still don't have a good speed, try decreasing the frame n/50 option at Options->Speed menu, and disable the flash option at Options menu;

. if by the contrary, you have too much speed, put frame n/50 option at Options-Speed menu to 1, and if that doesn't still help you, go to Options-> Speed menu and start increasing the Delay number;

. Double-clicking in a .SCR file will not do any good, since WSpecEm will reset itself upon boot;

. If you are using 16 colours, sometimes the subset of the palette chosen by Windows for WSpecEm is flawed. If this happens, you'll have to shutdown and boot Windows again;

. If a key becomes stuck, press it again. That should solve the problem. A key can became stuck if you press it down, select the a Windows menu and release it before leaving the menu;

. Sometimes, when you are using the extended keys, you can get only one key instead of the combination of them (for example when pressing DELETE you get 0). This is a Spectrum ROM related bug and was present on the real machines;

. If you have problems with SHIFT keys stuck trying to get the E cursor, use TAB instead to get the E cursor -- this problem is more likely to occur under Windows 95 than Windows 3.x;

. if you load a ROM, it will be used until you load another ROM on the emulator;

. If the emulator presents a message, he could not load a ROM that’s because the spectrum.rom file is not on the directory of WSpecem.exe file. Following that message, you can load your own ROM file;

. To load a .TAP or .BLK contents (binary image files of Spectrum tapes), do File->Open to the corresponding file, and then tape LOAD"" (j RIGTH-SHIFT-P RIGTH-SHIFT-P ENTER).

2.5. WING notes

- WinGDCs are NOT palette devices. You must change their colour tables using WinGSetDIBColorTable, not SelectPalette.

- WinGBitBlt and WinGStretchBlt only support blitting from WinGDCs to the screen.

- Using BitBlt and StretchBlt to blt from one WinGDC to another can be very slow when a clipping region has been selected into the destination.

2.6. Program history

25th February 97

. V1.30 released.

11th February 97

. some minor fixes relating to the fact the screen is no longer entirely updated all the time;

10th February 97

. included Walter Schroeder CreateDIBSection fix (and other minor fixes, relating w/ porting WSpecEm to 32-bits, which was mixed with the ongoing 32-bit work) - also a bug that prevented WinG and CreateDIBSection to produce incremental screen updates is now fixed (major speed improvement);

12th January 97

. setup script modified to allow disabling/enabling extensions associating under the file manager;

9th January 97

. z80_load() modified to allow reading v3.05 .Z80 files.

30th December

. R register is now updated in the ldir() and lddr() loop code.

. ld_a_r and ld_r_a() corrected to match exactly the Z80 processor. Now uspirits.tap runs.

. Added code to return 0 when reading port 0xFFFD, just to run uspirits.tap.

. neg() error corrected. Now Super Hang-on runs correctly.

. v1.21 released.

24th December

. Corrected bit_n_phl, thanks to a Blood's hint about 'Doods and the things' were crashing the emulated Spectrum after clicking the start option.

3rd October

. Executable compiled for 386 processor.

. v1.20 released.

2nd October

. Added ZX's .SNP snapshot format.

30th September

. Several non-critical Window-related bugs corrected.

. Started adding code for 32-bit operation.

19th September

. Created Windows help file.

18th September

. Corrected .BMP saving code in WSpecEm.c, with the help of my friend Jose Caetano Silva [jcaetano@idt.ipp.pt].

14th September

. Changed Speed Popup to Options menu.

13th September

. Created Windows'95 compliant setup with InstallShield SE.

3rd September

. Reorganised system menu.

. Added keyboard template and joystick control from Axle Quentin, who sent a modified WSpecEm version. Also, corrected it to allow the floating keyboard template to generate keypresses with a mouse.

1st September

Returned to Portugal. Finally, I am no longer forbidden to include other's people code and have a proper compiler! [I could not compile some code sent to me in the university campus];

. SEM snapshot format appended to the emulator-recognised formats. As found out, it appears it doesn't save the interrupt flags, as apparently in SpecEmu, each time the menu has control, they are disabled. Someone,

comments?

. OUTI and OUTD (inout.c) did nothing. Pointed out by Gilad Raz;

. 20ms constant timing var in wpsecem.c was short instead of long,
so automatic speed control was there from the beginning, but didn't work.
Pointed again by Gilad Raz;

. Changed the timing check to just before the execute() call. This way
it will be more accurate.

1st August

. Fixed special case of C flag handling in add_hl_ss and adc_a_r
adc_a_pss. Now CIRCLE basic instruction and any games that call the ROM
circle subroutines works perfectly. Thanks, Blood.

. Released v1.13.

20th June

. From now on, the next file loaded after a .ach file, will reload
the last active ROM file.

. Released v1.12.

. Handed the project at the University of West of England, at
Bristol, UK.

13th June

. Fixed the overflow logic calculation (flag P), at MATH16BI.c, for instructions adc_hl_ss() and sbc_hl_ss() family. Now apolo11.z80 works. It was some glitch, because the 8 bit-overflow is well done. Thanks again, Blood.

6th June

. Corrected TAPE2TAP, it appeared to have some problems with 16 bit-soundcards and upper.

2nd June

. Implemented 4x4 size;

. Included on the menu the equivalent keys, as usually Windows programs do. Thanks for the hint, Cotrina.

. Released v1.11.

30th May

. Removed a bug where if you paused the emulator, loaded a snap and unpaused it again, you would be back to the name of the previous snapshot. Thanks go to Blood [L.D.Thonks@bra0202.wins.icl.co.uk].

29th May

. Updated document with development notes, relating to questions a few people did by email, and some new insights.

28th May

. Appended a document with tips for reading Spectrum tapes after receiving a mail from Italy asking me related questions.

6th May

. Appended (...) to Speed and Poke on the menu to conform to Windows menu notation;

. Modified the position of the video flashing instruction, to avoid an error where sometimes the last byte wrote to the pixel area would not show up.

25th May

. Screen generation modified -- instead of generating always 50 frames per second, now it's user definable. 25 will do perfectly, it was stupid and slow to do 25 -- anyway at Europe we have 24 at TV and nobody complains --- emulator is a lot more faster, now; did that after suggestions from Cotrina and Marat Fazullin;

. As now, we have a faster emulation, built a configurable delay option;

. When a pause is made, invalidate to screen to force Windows to paint it, so it will work with new frame time scheme.

. Included spectrum FAQ v3.0;

. V1.10 Released.

22nd May

. Tried to increase task priority, but that appears to be a feature of Windows 3.0 no longer working under 3.11, set WinMain() at wspecem.c for details;

. V1.03 Released.

. Poke option was recognising numbers as signed and did not work with numbers that exceed 32767. Thanks goes to Alvin [sealbrec@acs.ucalgary.ca];

V1.03.01 Released.

21st May

. Due to an error, the ghost keys were not working...

. Corrected issue 2 emulation, thanks to a warning from James McKay [com40014@paisley.ac.uk]. [Hauting Hedges didn't work];

. Now 5 on the numeric keypad is fire too, besides 0, when Sinclair joystick is active;

. group of instructions ld Iy,r and ld iY,r weren't present on array of functions instruc_tablIY;

. ld_iY_a() corrected. It loaded A on the HY register. American 3D Pool is now working [am3d.z80]. It displayed all the characters garbled on the screen. [The game was a hint from James];

. Date of release placed on the Windows about box;

. T-states of bit_r family of instructions corrected and 1 T-state of delay introduced when reading or writing the ULA. Thanks goes to Gerton Lunter for the file benchmrk.z80 and the Spectrum info[gerton@rcondw.rug.nl].

20th May

. Corrected more two errors: still had a problem dealing with a non-existent ROM at the program startup time, and got a loop when you typed LOAD "" without Loading in the menu, a .BLK or .TAP tape file;

. Increased stack size;

. Found another bug: when loading any kind of files, except snapshots, F was modified;

. Send now image dirty signals to windows after opening a load, saveas or about box;

. Corrected .TAP and .BLK file handling. If the block was not fully loaded (e.g. searching for blocks, problems occurred). Error spotted thanks to the 80 (header+body) tape blocks of AlchNews Issue 19;

. tap_load() modified to follow more closely the real ROM routine.

. v1.02 released

19th May

. .SLT file loading scheme slightly modified, to allow for future extensions of the format, again following Damien suggestions;

. After complaints from Cotrina implemented ghost keys (fake closures);

. Now it unpatches the ROM before saving it: it was trashing it in others versions in a way that it could not be used with other emulators in case it was saved by mistake;

. Corrected bug concerning .BLK tape files.

. v1.01 released

18th May

. PCX saving implemented.

16th May

. Now emulation can load properly .SLT files and it's levels, after complaints from Damien;

. v1.0 Released.

15th May

. Corrected it, didn't release memory when leaving;

. Sent v0.01.05b to Damien and Cotrina;

. Correct a few mistakes from the documentation;

. Included WSpeEm.ico designed by Damien Burke;

. Modified scheme names presentation on the windows caption, Model 3 to Issue 3 in the options menu and F5 to reset after Damien Burke suggestions;

. Forgot to modify version and sent it to Cotrina;

. Prevent saving a snapshot which extension is not recognised, following a complaint from Damien Burke;

. Released to the public.

14th May

. Corrected another error, where the caption box would not be modified for snapshots when icons were clicked or dragged and dropped into the application;

. Sent v0.01.04b to Damien and Cotrina.

13th May

. Corrected error where if ROM file was not found, the emulator was locked in a loop, relating to a dialog box [bug report: Damien Burke];

. Implemented F2 as saveas and F3 as load keys, following a suggestion from Damien Burke.

12th May:

. Now I also emulate issue 3, since Cotrina found Abu Simbel Profanation had problems with emulation. I found out that this game only works on issue 3 speccys. I didn't emulate issue 3, because I tough it was insane to ignore the issue 2 market;

. Include menu option issue 3/issue 2;

. Modified timings according to discussions with Ian Collier;

. Modified EI routine to prevent interrupts at the end of EI only the interrupts were disabled at the beginning;

. Correct bug where if you double-clicked an icon, the ROM file would not be opened;

11th May:

. Corrected sizing of window. It did only take in account one frame and not the two frames. Now the screen updates are indeed faster. It appears the program was being penalised for not being in a multiple of the screen resolution. [anyone who played with AVI files will understand this]. That's way it was so slow...and that's because I lost so much of may time speeding it up.

10th May:

. Corrected Pause and iconify buggy behaviour after a Cotrina bug report. The program was trying to paint the screen in the icon space, and Windows kept updating the icon, this happening ad eternal.

9th May:

. A kind of paint cache implement to improve speed of emulation;
. Poke interface implemented;

. v0.01.03b released [to Cotrina].

8th May:

. Flash implemented;

. Tried to implement HCR-like resolution with horrendous and too slow results;

. v0.01.02b released [to cotrina@lia01.unizar.es (Francisco Cotrina)].

7th May:

. Implemented .SCR format;

. Handle DI/HALT situation;

. TAP saving corrected (only MULTIFACE like, yet);

2nd May 96:

. Implemented .TAP format successfully;

30th April 96:

. Started implementing .SLT format;

25th April 96:

. Cleaned a bit the files. Started implementing .TAP format;

19th April 96:

. No more WM_TIMER messages: painting logic now is done at the end of the Spectrum virtual time. The screen appears now to not be so 'jerky';

18th April 96:

. ED/FD level loader trap implemented;
. v0.01.01b released [to D.M.Burke-CSSE94@cs.bham.ac.uk (Damien Burke)].

16th April 96:

. Cache implemented in SNALOAD.c --- snapshots load time greatly reduced;

. v0.01.00b released. [to deec45@tom.fe.up.pt(Filipe Silva) and deec322@tom.fe.up.pt(Paulo Augusto)];

13th April 96:

. Improved application menus.

12th April 96:

. Another error corrected: bits 6 and 7 of ULA port were always 0 instead of 1 [Rick Dangerous didn't run];

. Designed debugger screen;

. Investigated why Rick Dangerous does not work with a joystick, but it is a problem with the snapshot -- maybe it was loaded in an emulator with no joystick support.

11th April 96:

. Found several bad defined instructions in the instruction table. dec_iY, dec_Yi were both defined as dec_yi. [Navy Moves 2 crashed instead of asking for the codes].

10th April 96:

. Downloaded a few snapshots from Internet;
. Added support for longfilenames under Windows'95.

9th April 96:

. Sound added to emulation.

8th April 96:

. Replaced parity function at MISC.c by a parity table.

7th April 96:

. Finished modifications. The emulation appears to be notably faster.

6th April 96:

. Continued to modify emulation for a faster speed. Appended all the undocumented opcodes that never were implemented to date (the ones with the CB prefix).

5th April 96:

. Modified IX/IY logic;

. Modified handling of IX and IY prefixes -- extensive changes in the code. This will fix the error spotted and will increase once more the emulation speed;

. Modified ld_a_r() and ld_r_a() for a new handling of R register.

4th April 96:

. modified R register handling;

. corrected runaway condition in getbyte() with corrupted snapshots (would kill emulator);

. Modified central loop of emulation (execute()) for a faster emulation;

. Spotted an error in Z80 emulation. When handling IX or IY prefixes, if the next instruction was not a HL instruction, the prefix would affect all instructions until if found a HL instruction or an ED prefix. It's surprising how only failed 4 of nearly 2000 spectrum programs tested.the

. Added .SIT and .BLK handling logic

3. Program Interface

3.1. Windows menus

Option	Activity	Short-cut
Open	opens a snapshot or makes the emulator point to a tape file for later use;	Alt-Fo, F3
Save	Saves the current emulator state under the name of the current snapshot;	Alt-Fs
Save as	saves the current emulator state asking for a name first;	Alt-Fa, F2
Reload	reloads the current snapshot to memory;	Alt-Fr
Exit	exits the emulator.	Alt-Fx

Table C-1 File Menu.

Option	Activity	Short-cut
Sound	Enables or disables sound during the emulation;	Alt-Oo
Flash	Enable or disables flash; [4]	Alt-Of
Issue 3	Toggles between issue 2 and issue 3 emulation	Alt-Oi
1x1 size	Client area of emulator will be 256x192;	Alt-O1
2x2 size	Client area of emulator will be 256*2x192*2;	Alt-O2
3x3 size	Client area of emulator will be 256*3x192*3;	Alt-O3
4x4 size	Client area of emulator will be 256*4x192*4;	Alt-O4
Save	Save options.	Alt-Os

Table C-2 Options menu.

Option	Activity	Short-cut
Pause	Pauses / unpauses the emulator;	Alt-Mp, F4
Poke	Writes a byte on a specified address;	Alt-Mo
Speed	Customises the emulation speed;	Alt-Ms
Reset	Resets the emulated Z80;	Alt-Mr
NMI	Assert the NMI signal on the emulated Z80.	Alt-M

Table C-3 Misc menu.

Option	Activity	Short-cut
About	Application credits.	Alt-Ha.

Table C-4 Help Menu.

[4] Flash can be disabled to gain speed.

. kempston joystick as cursor keys with Fire as CTRL (if NUM LOCK off, the numeric keypad will work as a kempston joystick);

. Sinclair joystick as numeric keypad if NUM LOCK on (8 up, 2 down, 4 left, 6 right, 0 and 5 are both fire);

. short-cut load and save keys, a' la Z80 (F2 - save and F3 - load, F5 - reset);

. Key F4 is now pause;

. maps the Spectrum keyboard in the PC keyboard, with some extra pc-keys:

ESC : Caps-shif+SPACE (CTRL-BREAK in a Speccy)
BACKSPACE : same as Caps-shift+0 (DELETE in a Speccy)
LEFT SHIFT : same as Caps-shift
RIGHT SHIFT : same as Symbol-shift
TAB : same as Caps-shift + Symbol-shift (Cursor E)

+,-,*,/ extended keys, operate as their Spectrum Plus counterparts.

4. Selected snapshots

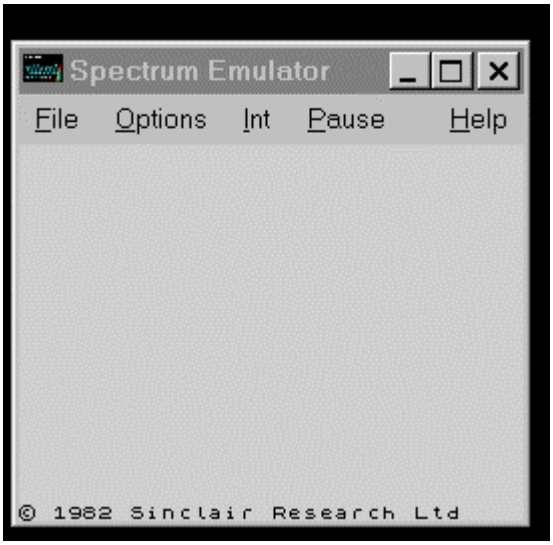

Figure C-4-1 WSpecEm just after booting.

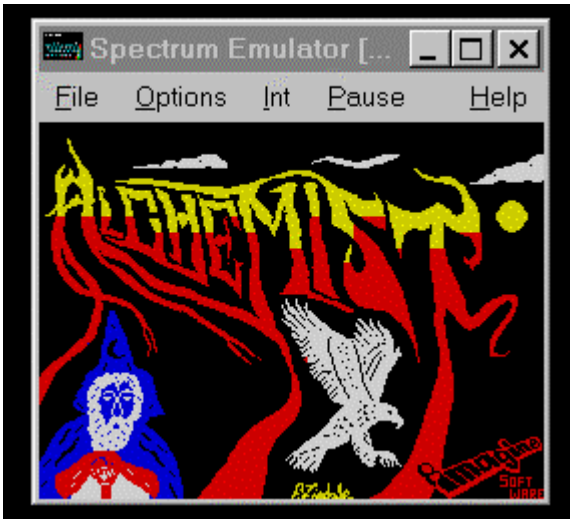

Figure C-4-2 Running Alchemist.

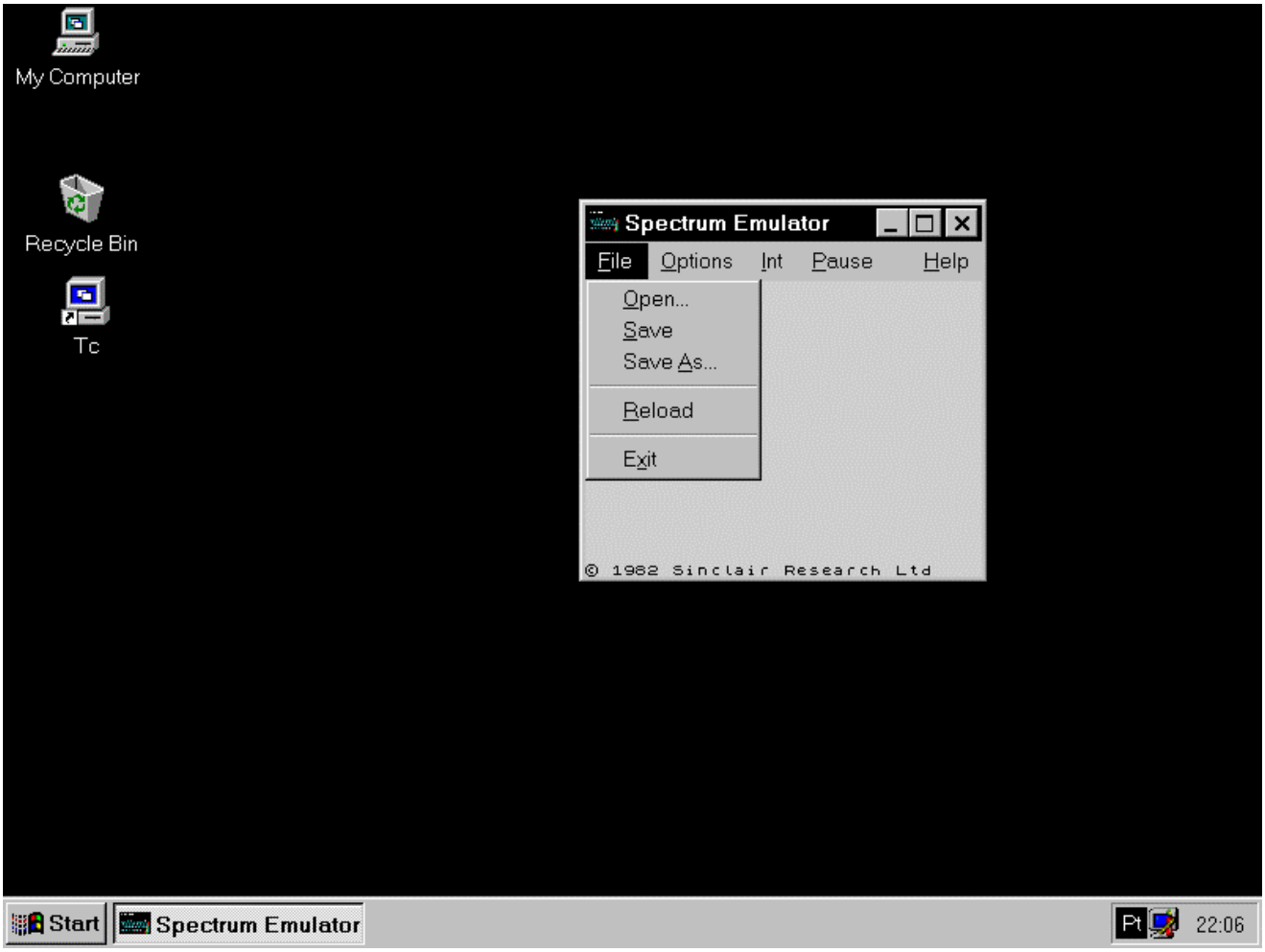

Figure C-4-3 Showing the file menu.

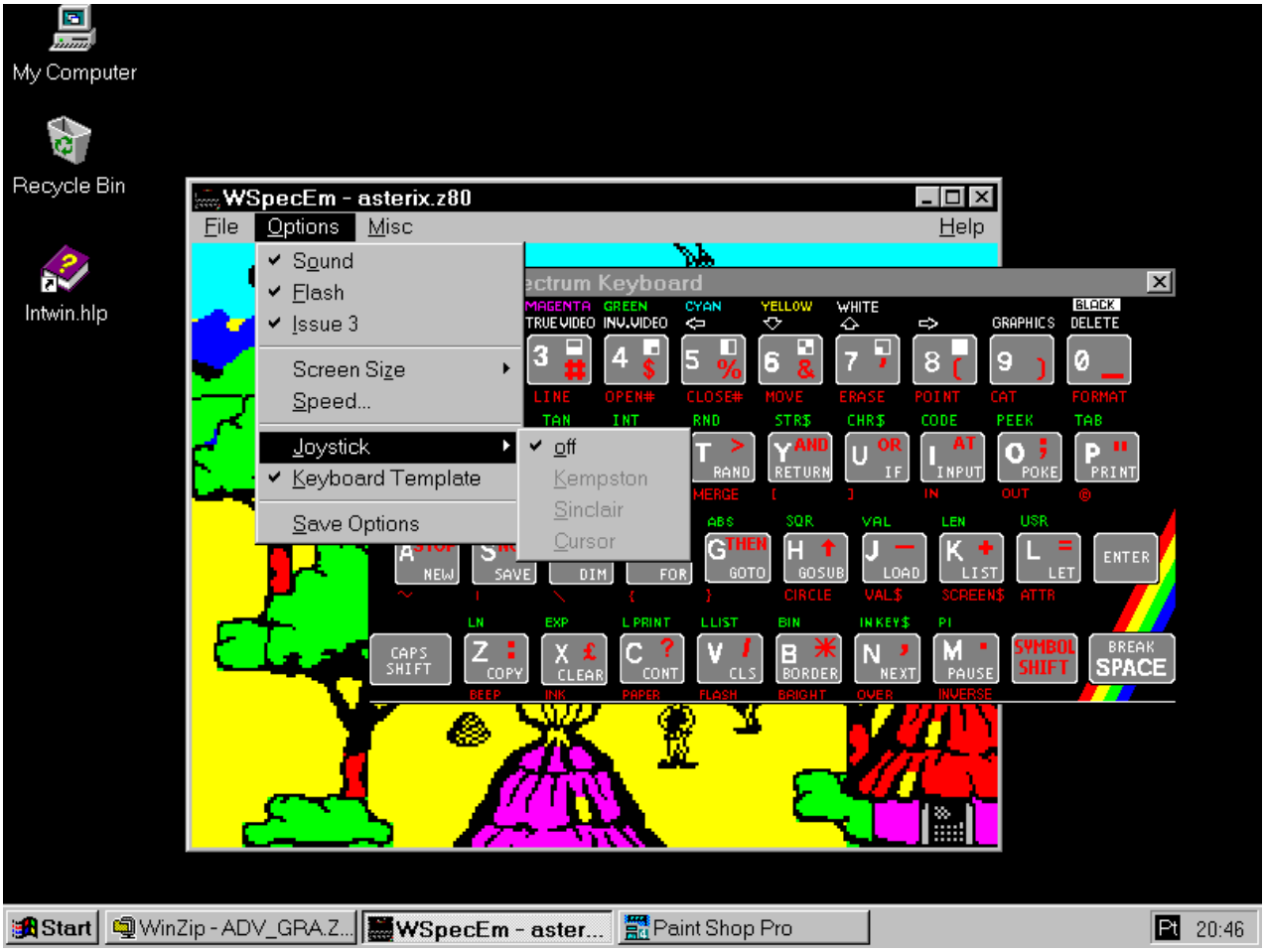

Figure C-4-4 Showing the options menu.

Figure C.4 5 The Misc Menu

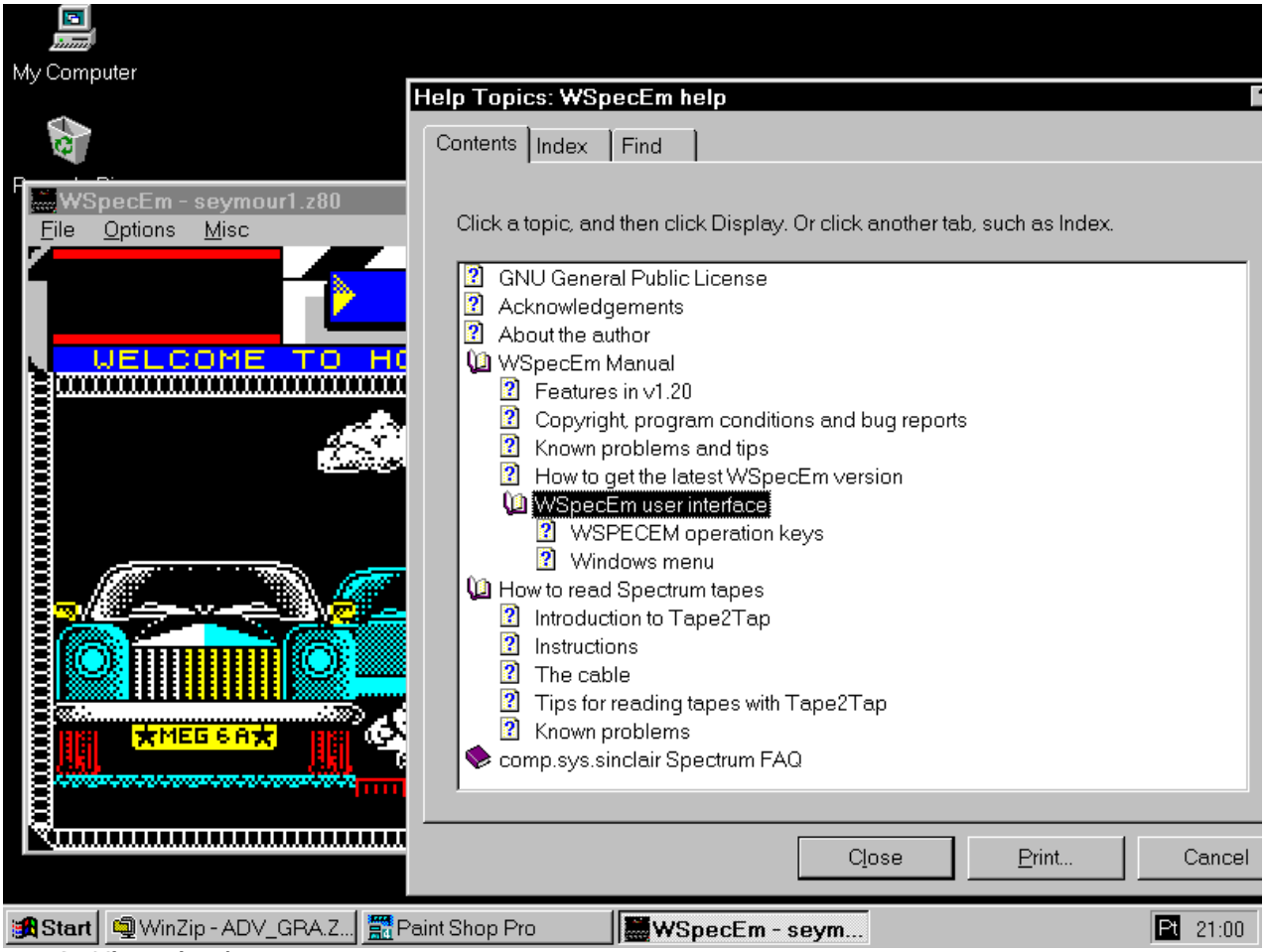

Figure C-4-6 Showing the Help Engine.

5. CD Contents

_st__ _o cofs22A CD and a diskette have been handed enclosed with this dissertation in Britain. In Portugal, the CD has helped creating a spectrum site at ftp.idt.ipp.pt. Here the contents of this CD are being exposed, for the diskette contents, please see the file README.TXT in its main directory. If you only want to use the diskette, it's enough to type a:setup. The user will also have to copy manually the WinG driver files to his default \Windows\system directory. Setup does all the work, including installing the program, creating the icons, and the file associations.

\emul

wing10.zip	- WinG package from Microsoft
mylinks.htm	- Rui's Internet links

\emul\others - Non-Spectrum emulators and related material

\emul\others\docs - Emulation theory and emulators documents

\emul\others\emu - Dozens of several non-Spectrum emulators, including Executor

\emul\others\sources - Non-Spectrum emulator sources

\EMUL\SPECTRUM - Spectrum related material root directory

\EMUL\SPECTRUM\DOCS - Documents ranging from Spectrum HTML pages, to Z80
texts, hardware, news postings... relevant material, part of it used in the project making

\EMUL\SPECTRUM\EMU - Spectrum emulators, for several platforms

\EMUL\SPECTRUM\PICS - Several megabytes of Speccy related-pictures, from game
covers to the Spectrum board scheme

\EMUL\SPECTRUM\ROMS - Spectrum ROMs

\EMUL\SPECTRUM\SNAPS - Spectrum Snapshots

When there's a TAP directory inside a named directory, it's games of that type in game format. 128 directory, games for Spectrum 128. Other directory name, it's the game in DAT format.

\EMUL\SPECTRUM\SNAPS\ADV_GRA - Graphical adventures

\EMUL\SPECTRUM\SNAPS\ADV_TXT - Text adventures

\EMUL\SPECTRUM\SNAPS\BAD - Corrupted images

\EMUL\SPECTRUM\SNAPS\CODES - programs with codes

\EMUL\SPECTRUM\SNAPS\DEMOS - Demos

\EMUL\SPECTRUM\SNAPS\EDU - Education

\EMUL\SPECTRUM\SNAPS\GAMES - Miscellaneous games

\EMUL\SPECTRUM\SNAPS\INCOMPLE - Incomplete games

\EMUL\SPECTRUM\SNAPS\MAZES - Mazes

\EMUL\SPECTRUM\SNAPS\MULTI - Misc, multilevel

\EMUL\SPECTRUM\SNAPS\PLATFORM - Platform games

\EMUL\SPECTRUM\SNAPS\REFLEX - Addictive games

\EMUL\SPECTRUM\SNAPS\SHOOTMUP - Shoot'em Up

\EMUL\SPECTRUM\SNAPS\SIMUL - Simulators

\EMUL\SPECTRUM\SNAPS\SPORTS - Sports

\EMUL\SPECTRUM\SNAPS\STRATEGY - Strategy

\EMUL\SPECTRUM\SNAPS\TURBO - Tape files with Turbo Loader routines

\EMUL\SPECTRUM\SNAPS\UTILS - Utilitarian programs (compilers, assemblers, copy programs, spreadsheets, Electronic Magazines

\EMUL\SPECTRUM\SNAPS\VOCS - Vocs with tapes sampled

\EMUL\SPECTRUM\SNAPS\ZX81 - ZX81 Software

\EMUL\SPECTRUM\SOURCES - Spectrum related sources

ZX82_ROM.TXT - Spectrum ROM

\EMUL\SPECTRUM\SOURCES\CONVERT - Sources to convert between several formats

\EMUL\SPECTRUM\SOURCES\DOS - Sources for emulators under DOS

EKIT11 - C
JPPS0608 - Assembly
X128 - C
Z80V201 - Assembly

\EMUL\SPECTRUM\SOURCES\MAC - Sources for emulators under MAC OS

\EMUL\SPECTRUM\SOURCES\WINDOWS- Sources for WSpecEm as Start June’96

\EMUL\SPECTRUM\SOURCES\X - Sources for emulators under X-Windows

X128_0_3 - C
X128_0_4 - C
XZ80 - C
ZXZ-1012 - C

Appendix D

1. Relevant sources

In this appendix, it was printed a few modules relevant by their nature for this project.

1.1. WSpecEm.c

This is the main windows interface, and where it's really implemented almost all the interface between the emulation, Windows and the user. As such is one of the most important modules in the project.

```
/* WSpecem.c : Windows interface for WSpecem emulator.
 *
 * Copyright 1996 Rui Fernando Ferreira Ribeiro.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#define APPNAME "SpecEmulApp"


// main window client area size
#define X_CORD 256
#define Y_CORD 192

/*
 *    WSpecEm.C
 *
 */

#include <windows.h>    // Defines bulk of Windows functions and such...
#include <windowsx.h>
#include <mmsystem.h>   // Defines additional Multi-Media functions...
#include <shellapi.h>   // Drag-and-drop
#include <string.h>
#include "env.h"
```

```
#include "snd/wave.h"

#if !defined(WIN32)
   /*#include "c:/wing/include/wing.h"*/ // Defines WinG functions. Not use on Win32
   #include "wing.h"
#endif

#include <stdlib.h>
#include "wspecem.h"            // Local header

//  ========================================================================
//  GLOBAL VARIABLES ------------------------------------------------------
//  ========================================================================

extern unsigned char keybd_buff[8]; /* Spectrum key states */
extern unsigned char joystick;      /* Joystick positions  */

// callback function to interface
extern BOOL CALLBACK DoPoke(HWND,UINT,WPARAM,LPARAM);
extern BOOL CALLBACK DoSpeed(HWND,UINT,WPARAM,LPARAM);

char    szAppName[]= APPNAME;   // A handy string to identify this app
char    szTitle[] = "Spectrum Emulator"; // For the title bar

HINSTANCE hInstApp;     // A handle that identifies this 'process'
HWND      hwndApp;      // A handle that identifies the main window
HPALETTE  hpalApp;      // A handle that identifies the main palette
BOOL      fAppActive;   // A boolean that refers to this app being 'foreground'

// var to track state paused/not paused
static BOOL NotPaused = 1;

// Define a structure that we will be using for keeping information about the
// image we are currently drawing into.
typedef struct _IMAGE {
      BITMAPINFOHEADER bi;  // Bitmap header information.
      RGBQUAD aColors[256]; // Palette color table
      union { // Now for the pointer to the data buffer we can whack on:
            LPVOID  lpvData; // This is the type that WinG likes to deal with
            LPBYTE  lpIndex; // This is just to make it easier for us to access it
      } data;
} _IMAGE;
_IMAGE image; // Contains most necessary information about the image to display


// Define a structure for holding our palette data. This is the color palette
// that will be assigned to both the above image, and be selected into the
// 'Device Context' for the screen so we will have an 'Identity' palette which
// will make for much faster screen updating.
typedef struct _PALETTE
{
      WORD Version;
      WORD NumberOfEntries;
      PALETTEENTRY aEntries[256];
} _PALETTE;
_PALETTE LogicalPalette = {0x300, 256}; // The 'logical' palette we will use
                                 // "0x300" = Windows 3.0 or later
                                 // "256" = Number of colors

long Orientation = 1; // Bitmap Orientation: TopDown=1, BottomUp=-1
HDC hdcImage = NULL;  // A handle to the Device Context for our image

HBITMAP gbmOldMonoBitmap = 0;   // Storage for the 'original' bitmap from our
                          // Device Context, we need to restore it
                          // later on, so we need to save it here.

/* RGB 'Spectrum' colors */
static unsigned short rgbvals[16][3]={
               /* Normal colours */
              { 0x00, 0x00, 0x00}, { 0x00, 0x00, 0xcf},
              { 0xcf, 0x00, 0x00}, { 0xcf, 0x00, 0xcf},
              { 0x00, 0xcf, 0x00}, { 0x00, 0xcf, 0xcf},
```

```
                { 0xcf, 0xcf, 0x00}, { 0xcf, 0xcf, 0xcf},

                /* Brigth colours */
                { 0x00, 0x00, 0x00}, { 0x00, 0x00, 0xff},
                { 0xff, 0x00, 0x00}, { 0xff, 0x00, 0xff},
                { 0x00, 0xff, 0x00}, { 0x00, 0xff, 0xff},
                { 0xff, 0xff, 0x00}, { 0xff, 0xff, 0xff}
};

unsigned char ChangeFlashTime = 0; /* count time till inverting colours */

//  ========================================================================
//  FUNCTION DEFINITIONS ------------------------------------------------
//  ========================================================================

// Forward declarations for all functions.
// Listed in order they appear in this source listing

BOOL AppInit(HINSTANCE hInst,HINSTANCE hPrev,int sw);
LONG FAR PASCAL AppWndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam);
BOOL AppPaint (HWND hwnd, HDC hdc);
LONG AppCommand (HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam);
BOOL FAR PASCAL AppAbout(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam);
void AppExit(void);


/*--------------------------------------------------------------------------*\
|   WinMain( hInst, hPrev, lpszCmdLine, cmdShow )                            |
|                                                                            |
|   Description:                                                             |
|       The main procedure for the App.  After initializing, it just goes    |
|       into a message-processing loop until it gets a WM_QUIT message       |
|       (meaning the app was closed).                                        |
|                                                                            |
|   Arguments:                                                               |
|       hInst           instance handle of this instance of the app          |
|       hPrev           instance handle of previous instance, NULL if first  |
|       szCmdLine       ->null-terminated command line                      |
|       cmdShow         specifies how the window is initially displayed      |
|                                                                            |
|   Returns:                                                                 |
|       The exit code as specified in the WM_QUIT message.                   |
|                                                                            |
\*--------------------------------------------------------------------------*/
int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPrev, LPSTR szCmdLine, int sw)
{
   MSG     msg;

   // NOTE: On Win32, hPrev will -always- be NULL

   // Do application initialization stuff
   if (!AppInit(hInst,hPrev,sw)) {
      return 0; // Something failed to initialize
   }

   init_emul(hInst);

   /* If there is a snapshot in the command line, it is opened */
   if(szCmdLine[0])
      open_sna(szCmdLine);

#if !defined(WIN32)
   /*{
   WORD (FAR  PASCAL *SetPriority)(WORD, WORD);

   SetPriority = GetProcAddress(GetModuleHandle("KERNEL"), "SetPriority");
   if(SetPriority != NULL) */
                        /* -32, 15 -- numbers below 0 are locking system */
   /*    SetPriority(GetCurrentTask(), (WORD)0);
   else
      MessageBox(NULL, "!!!", "Not worked", MB_ICONHAND);
   } */
```

```
#endif


   // Poll for messages from event queue.
   // loop continues until WM_QUIT is encountered
   for (;;) {
      if (PeekMessage(&msg, NULL, 0, 0,PM_REMOVE)) {
        if (msg.message == WM_QUIT) {
           // When WM_QUIT comes through, we're DONE!
           break;
           }
        TranslateMessage(&msg); // Messages for menu keys
        DispatchMessage(&msg);
        }
      else
      {
         if(NotPaused)
           execute();

        if(!IsIconic(hwndApp))
        {
           static char FrameCounter = 0;
#if defined(WINDOWS_SOUND)
            static unsigned short freq;
#endif
           static unsigned short mscStart = 0;

           // if window needs a update
           if(WindowDirty)
           {
              if(FrameCounter++ == ScreenUpdate)
              {
                 FrameCounter = 0;
                 // Just force a redraw
                 InvalidateRect (hwndApp, NULL, FALSE);
               }
            }


            if(NotPaused)
            {
               if(++ChangeFlashTime == 100)
               {
                  // if it's time to invert colours...
                 ChangeFlashTime = 0; // reset counter
                 FlashState ^= 1;     // signal colours inverted
                 WindowDirty = 1;     // force a window redraw
                   if(FrameCounter > 1)
                   FrameCounter = ScreenUpdate-2;
                }

#if defined(WINDOWS_SOUND)
               /* Can anyone put Windows .WAV sound to work?
                */
               freq = do_int_tasks();
#endif
               /* Watch for 20ms */
               while((timeGetTime() - mscStart) < 20);
#if defined(WINDOWS_SOUND)
               /* More bits needed to Windows .WAV sound... */
               StopSnd();
               FPlaySnd((float)freq, -1, 11025, 8, 1);
#endif
               mscStart = timeGetTime();
            }
         } /* Iconic */
         else
             /* if program is in icon form and not paused, pause it */
           if(NotPaused)
           {
            PostMessage(hwndApp, WM_COMMAND, IDM_PAUSE, 0L);
           }
```

```
      }
      }
      AppExit();        // Do application exiting stuff

      return msg.wParam;
}

/*--------------------------------------------------------------------------*\
|   AppInit( hInst, hPrev)                                                   |
|                                                                            |
|   Description:                                                             |
|       This is called when the application is first loaded into             |
|       memory.  It performs all initialization that doesn't need to be done |
|       once per instance.                                                   |
|                                                                            |
|   Arguments:                                                               |
|       hInstance       instance handle of current instance                  |
|       hPrev           instance handle of previous instance                 |
|       sw              window showmode                                      |
|                                                                            |
|   Returns:                                                                 |
|       TRUE if successful, FALSE if not                                     |
|                                                                            |
\*--------------------------------------------------------------------------*/
BOOL AppInit(HINSTANCE hInst,HINSTANCE hPrev,int sw)
{
   WNDCLASS cls;
   char szBuf[260];

   // read options
   GetWindowsDirectory((LPSTR)szBuf, 259);
   strcat(szBuf, "\\wspecem.ini");
   open_sna((LPSTR)szBuf);

   // Save instance handle for DialogBoxs as a global variable
   hInstApp = hInst;

   if (!hPrev) {
      // We don't already have a version running, so we need to
      // register our window class with the system
      cls.hCursor        = LoadCursor(NULL,IDC_ARROW);
      cls.hIcon          = LoadIcon(hInst,szAppName);
      cls.lpszMenuName   = szAppName;
      cls.lpszClassName  = szAppName;
      cls.hbrBackground  = (HBRUSH)(COLOR_WINDOW + 1);
      cls.hInstance      = hInst;
      cls.style          = CS_BYTEALIGNCLIENT | CS_VREDRAW
                     | CS_HREDRAW | CS_DBLCLKS;
      cls.lpfnWndProc    = (WNDPROC)AppWndProc;
      cls.cbWndExtra     = 0;
      cls.cbClsExtra     = 0;

      if (!RegisterClass(&cls)) {
        return FALSE; // Failed to register class. No need to continue.
      }
   }
   hwndApp = CreateWindow (szAppName,           // Class name
                       szTitle,             // Caption
                       WS_OVERLAPPEDWINDOW, // Style bits
                       CW_USEDEFAULT, 0,    // Position (x,y)
                       // Size (w,h)
                       X_CORD*Scale+GetSystemMetrics(SM_CYFRAME)*2,
                       Y_CORD*Scale+GetSystemMetrics(SM_CXFRAME)*2+
                            GetSystemMetrics(SM_CYMENU) +
                            GetSystemMetrics(SM_CYCAPTION),
                            (HWND)NULL,    // Parent window (no parent)
                            (HMENU)NULL,   // use class menu
                            hInst,         // handle to current process instance
                            (LPSTR)NULL    // no params to pass on
                       );

   if (hwndApp) {
```

```
      ShowWindow(hwndApp,sw); // Time to display the window.
      return TRUE;
      }
   else
      {
      return FALSE; // Failed to create window. No need to continue
      }
}

/*--------------------------------------------------------------------------*\
|   Resize( )                                                                |
|                                                                            |
|   Description:                                                             |
|       This is called when the main window is about to be resized.          |
|                                                                            |
|                                                                            |
|   Returns:                                                                 |
|       TRUE if successful, FALSE if not                                     |
|                                                                            |
\*--------------------------------------------------------------------------*/
BOOL ResizeWindow(void)
{
   HWND hwndAppOld = hwndApp;

   /* It's important destroying the window only after we have created
     a new one, or else we'll lose messages
    */
   hwndApp = CreateWindow (szAppName,           // Class name
                       szTitle,             // Caption
                       WS_OVERLAPPEDWINDOW, // Style bits
                       CW_USEDEFAULT, 0,    // Position (x,y)
                       X_CORD*Scale+GetSystemMetrics(SM_CYFRAME)*2,// Size (w, h)
                       Y_CORD*Scale+GetSystemMetrics(SM_CXFRAME)*2+
                            GetSystemMetrics(SM_CYMENU) +
                            GetSystemMetrics(SM_CYCAPTION),
                            (HWND)NULL,    // Parent window (no parent)
                            (HMENU)NULL,   // use class menu
                            hInstApp,      // handle to current process instance
                            (LPSTR)NULL    // no params to pass on
                       );

   if (hwndApp) {
      DestroyWindow(hwndAppOld);
      ShowWindow(hwndApp,SW_SHOW); // Time to display the window.
      return TRUE;
      }
   else
      {
      return FALSE; // Failed to create window. No need to continue
      }
}

/*--------------------------------------------------------------------------*\
|   AppWndProc( hwnd, uiMessage, wParam, lParam )                            |
|                                                                            |
|   Description:                                                             |
|       The window proc for the app's main (tiled) window.  This processes all |
|       of the parent window's messages.                                     |
|                                                                            |
\*--------------------------------------------------------------------------*/
LONG FAR PASCAL AppWndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam)
{
   //static int XOffset, YOffset;
   static int dxClient, dyClient; // The 'Client' size. (drawing area of window)
   PAINTSTRUCT ps;
   HDC hdc;
   UINT uMappedColors;
   HMENU hmenu;

   switch (msg) {
      case WM_CREATE:
        // This occures during 'CreateWindow' time. Here is where we can
```

```
        // easily initialize some things for this window.
        srand ((int)GetTickCount()); // initialize the random seed
        //SetTimer(hwnd, 1, 1, NULL);  // start up the WM_TIMER messages

        DragAcceptFiles( hwnd , TRUE );

#if defined(WINDOWS_SOUND)
        /* Init wave driver */
        FInitSnd(); /* if true, init sucessfull*/
#endif

        // Init options on the menu
        hmenu = GetMenu(hwnd);
        CheckMenuItem(hmenu, IDM_SIZE1,  (Scale==1)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_SIZE2,  (Scale==2)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_SIZE3,  (Scale==3)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_SIZE4,  (Scale==4)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_SOUND,  (bSoundOn)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_COLOUR, (bFlashOn)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_MODEL3, (bModel3)?MF_CHECKED:MF_UNCHECKED);
        CheckMenuItem(hmenu, IDM_DEBUG, MF_UNCHECKED);
        break;

       //case WM_TIMER:
       //         break;

       case WM_ACTIVATEAPP:
         // The application z-ordering has changed. If we are now the
         // foreground application wParm will be TRUE.
         fAppActive = (BOOL)wParam;
         break;

      case WM_ERASEBKGND:
        return TRUE;
        /* break; */

      case WM_SIZE:
        // This message comes to us because the window size has been
        // changed. It also comes to us when the application is first
        // being executed.
        dxClient = LOWORD(lParam); // The 'new' width of our window
        dyClient = HIWORD(lParam); // The 'new' height of our window

        if(!hdcImage)   {
           //  Create a new WinGDC and 8-bit WinGBitmap
           HBITMAP hbm;
           int Counter;
           HDC Screen;
           //RGBQUAD far *pColorTable;

           //  Get WinG to recommend the fastest DIB format
#if defined(WIN32)
           if (FALSE) {
#else
              if(WinGRecommendDIBFormat((BITMAPINFO far *)&image.bi)) {
#endif
                //  make sure it's 8bpp and remember the orientation
                image.bi.biBitCount = 8;
                image.bi.biCompression = BI_RGB;
                Orientation = -1;
                }
              else {
                //  set it up ourselves
                image.bi.biSize = sizeof(BITMAPINFOHEADER);
                image.bi.biPlanes = 1;
                image.bi.biBitCount = 8;
                image.bi.biCompression = BI_RGB;
                image.bi.biSizeImage = 0;
                image.bi.biClrUsed = 0;
                image.bi.biClrImportant = 0;
                }
```

```
image.bi.biWidth = X_CORD;
image.bi.biHeight = Y_CORD * Orientation;

//  create an identity palette from the DIB's color table

// Get the Device Context of the screen
Screen = GetDC(HWND_DESKTOP);


// Get the 20 system colors as PALETTEENTRIES
GetSystemPaletteEntries(Screen,0,10,LogicalPalette.aEntries);
GetSystemPaletteEntries(Screen,246,10,LogicalPalette.aEntries
      + 246);

// Only a few DCs available, free this up so we aren't a hog
ReleaseDC(0,Screen);

// Initialize the logical palette and DIB color table
// Note that we are doing this as double entries. Making
// sure that we keep both tables -identical- this is to
// make sure that we can end up with an 'identity palette'
// which means that both the colortable assigned to the DIB
// and the palette entries associated with the palette
// that is selected into the Device Context are identical.
for(Counter = 0; Counter < 10; Counter++) {
  // copy the system colors into the DIB header
  // WinG will do this in WinGRecommendDIBFormat,
  // but it may have failed above so do it here anyway

  // The low end colors...
  image.aColors[Counter].rgbRed =
             LogicalPalette.aEntries[Counter].peRed;
  image.aColors[Counter].rgbGreen =
             LogicalPalette.aEntries[Counter].peGreen;
  image.aColors[Counter].rgbBlue =
             LogicalPalette.aEntries[Counter].peBlue;
  image.aColors[Counter].rgbReserved = 0;
  LogicalPalette.aEntries[Counter].peFlags = 0;

  // And the high end colors...
  image.aColors[Counter + 246].rgbRed =
       LogicalPalette.aEntries[Counter + 246].peRed;
  image.aColors[Counter + 246].rgbGreen =
       LogicalPalette.aEntries[Counter + 246].peGreen;
  image.aColors[Counter + 246].rgbBlue =
             LogicalPalette.aEntries[Counter + 246].peBlue;
  image.aColors[Counter + 246].rgbReserved = 0;
  LogicalPalette.aEntries[Counter + 246].peFlags = 0;
  }

// Now fill in all of the colors in the middle to reflect
// the colors that we are wanting.
for(Counter = 10;Counter < (10+16);Counter++) {
  image.aColors[Counter].rgbRed =
       LogicalPalette.aEntries[Counter].peRed =
             rgbvals[Counter-10][0];
  image.aColors[Counter].rgbGreen =
       LogicalPalette.aEntries[Counter].peGreen =
             rgbvals[Counter-10][1];
  image.aColors[Counter].rgbBlue =
             LogicalPalette.aEntries[Counter].peBlue =
             rgbvals[Counter-10][2];
  image.aColors[Counter].rgbReserved = 0;

  // In order for this to be an identity palette, it is
  // important that we not only get this color, but that
  // we get it in THIS location. Using PC_NOCOLLAPSE tells
  // the system not to 'collapse' this entry to another
  // palette entry that already has this color.
  LogicalPalette.aEntries[Counter].peFlags = PC_NOCOLLAPSE;
  }
```

```
              // The logical palette table is fully initialized.
              // All we have to do now, is create it.
              hpalApp = CreatePalette((LOGPALETTE far *)&LogicalPalette);

              //  Create a WinGDC and Bitmap, then select away
#if defined (WIN32)
              // Create a DC compatible with current screen
              hdcImage = CreateCompatibleDC (NULL);
#else
              hdcImage = WinGCreateDC();
#endif
              image.bi.biWidth = X_CORD;
              image.bi.biHeight = Y_CORD * Orientation;

#if defined (WIN32)
             hbm = CreateDIBSection (hdcImage, (BITMAPINFO far *)&image.bi,
DIB_PAL_COLORS, &image.lpvData, NULL, 0);
#else
             hbm = WinGCreateBitmap(hdcImage,(BITMAPINFO far *)&image.bi,
&image.data.lpvData);
#endif
              // Make sure that 'biSizeImage' reflects the
              // size of the bitmap data.
              image.bi.biSizeImage = (image.bi.biWidth * image.bi.biHeight);
              image.bi.biSizeImage *= Orientation;
              //  Store the old hbitmap to select back in before deleting
              gbmOldMonoBitmap = (HBITMAP)SelectObject(hdcImage, hbm);
              PatBlt(hdcImage, 0,0,dxClient,dyClient, BLACKNESS);
              }
           break;

       case WM_COMMAND:
         return AppCommand(hwnd, msg, wParam, lParam);

       case WM_PALETTECHANGED:
         if ((HWND)wParam == hwnd) {
            break;
            }

         // fall through to WM_QUERYNEWPALETTE

        case WM_QUERYNEWPALETTE:
           hdc = GetDC(hwnd);

           if (hpalApp) {
              SelectPalette(hdc, hpalApp, FALSE);
              }

           uMappedColors = RealizePalette(hdc);
           ReleaseDC(hwnd,hdc);

           if (uMappedColors>0) {
              InvalidateRect(hwnd,NULL,TRUE);
              return TRUE;
              }
           else
              {
              return FALSE;
              }
           /* break; */


       case WM_PAINT:

         if(NotPaused && bFlashOn)
            do_flash(); // Spectrum specific

         // hack for flushing byte buffer
         writebyte(0x4000, readbyte(0x4000) );

         // Update main window
         hdc = BeginPaint(hwnd,&ps);
```

```
  SelectPalette(hdc, hpalApp, FALSE);
  RealizePalette(hdc);
  AppPaint (hwnd,hdc);
  EndPaint(hwnd,&ps);
  return 0L;

 case WM_KEYDOWN:
  /* map PC keys in the array that simulates the
     Spectrum keyboard (key down)
   */
  switch(wParam)
     {
     case '1':      keybd_buff[3] |= ~0xFE; break;
     case '2':      keybd_buff[3] |= ~0xFD; break;
     case '3':      keybd_buff[3] |= ~0xFB; break;
     case '4':      keybd_buff[3] |= ~0xF7; break;
     case '5':      keybd_buff[3] |= ~0xEF; break;
     case 'Q':      keybd_buff[2] |= ~0xFE; break;
     case 'W':      keybd_buff[2] |= ~0xFD; break;
     case 'E':      keybd_buff[2] |= ~0xFB; break;
     case 'R':      keybd_buff[2] |= ~0xF7; break;
     case 'T':      keybd_buff[2] |= ~0xEF; break;
     case 'A':      keybd_buff[1] |= ~0xFE; break;
     case 'S':      keybd_buff[1] |= ~0xFD; break;
     case 'D':      keybd_buff[1] |= ~0xFB; break;
     case 'F':      keybd_buff[1] |= ~0xF7; break;
     case 'G':      keybd_buff[1] |= ~0xEF; break;
     case VK_SHIFT:
                 if(((lParam >> 16) & 0x7F) == 0x2A)
                    keybd_buff[0] |= ~0xFE; /* CAPS SHIFT */
                 else
                    keybd_buff[7] |= ~0xFD; /* SYMBOL SHIFT */
                 break;
     case 'Z':  keybd_buff[0] |= ~0xFD; break;
     case 'X':  keybd_buff[0] |= ~0xFB; break;
     case 'C':  keybd_buff[0] |= ~0xF7; break;
     case VK_DIVIDE:
               keybd_buff[7] |= ~0xFD;
     case 'V':  keybd_buff[0] |= ~0xEF; break;
     case '0':  keybd_buff[4] |= ~0xFE; break;
     case '9':  keybd_buff[4] |= ~0xFD; break;
     case '8':  keybd_buff[4] |= ~0xFB; break;
     case '7':  keybd_buff[4] |= ~0xF7; break;
     case '6':  keybd_buff[4] |= ~0xEF; break;
     case 'P':  keybd_buff[5] |= ~0xFE; break;
     case 'O':  keybd_buff[5] |= ~0xFD; break;
     case 'I':  keybd_buff[5] |= ~0xFB; break;
     case 'U':  keybd_buff[5] |= ~0xF7; break;
     case 'Y':  keybd_buff[5] |= ~0xEF; break;
     case VK_RETURN: keybd_buff[6] |= ~0xFE; break;
     case 'L':  keybd_buff[6] |= ~0xFD; break;
     case VK_ADD:
               keybd_buff[7] |= ~0xFD;
     case 'K':  keybd_buff[6] |= ~0xFB; break;
     case VK_SUBTRACT:
               keybd_buff[7] |= ~0xFD;
     case 'J':  keybd_buff[6] |= ~0xF7; break;
     case 'H':  keybd_buff[6] |= ~0xEF; break;

     case VK_ESCAPE:
               keybd_buff[0] |= ~0xFE; /* CAPS SHIFT */

     case VK_SPACE:      keybd_buff[7] |= ~0xFE; break;
     case 'M':  keybd_buff[7] |= ~0xFB; break;
     case 'N':  keybd_buff[7] |= ~0xF7; break;
     case VK_MULTIPLY:
               keybd_buff[7] |= ~0xFD;
     case 'B':  keybd_buff[7] |= ~0xEF; break;


     /* Special keys */
     case VK_TAB: keybd_buff[0] |= ~0xFE;
```

```
                keybd_buff[7] |= ~0xFD;
                break;

        case VK_BACK: keybd_buff[0] |= ~0xFE; /* CAPS SHIFT */
                   keybd_buff[4] |= ~0xFE;
                   break;

        /* kempston joystick */
        case VK_LEFT:  joystick |= 2; break;
        case VK_RIGHT: joystick |= 1; break;
        case VK_UP:    joystick |= 8; break;
        case VK_DOWN:  joystick |= 4; break;
        case VK_CONTROL: joystick |= 16; break;
        /* Sinclair joystick */
        case VK_NUMPAD5:
        case VK_NUMPAD0: keybd_buff[0] |= ~0xFE;
                      keybd_buff[4] |= ~0xFE; /* 0 - fire  */
                      break;
        case VK_NUMPAD4: keybd_buff[0] |= ~0xFE;
                      keybd_buff[3] |= ~0xEF; /* 5 - left  */
                      break;
        case VK_NUMPAD6: keybd_buff[0] |= ~0xFE;
                      keybd_buff[4] |= ~0xFB; /* 8 - right */
                      break;
        case VK_NUMPAD8: keybd_buff[0] |= ~0xFE;
                      keybd_buff[4] |= ~0xF7; /* 7 - up    */
                      break;
        case VK_NUMPAD2: keybd_buff[0] |= ~0xFE;
                      keybd_buff[4] |= ~0xEF; /* 6 - down  */
                      break;
        }
        return 0L;

case WM_KEYUP:
        /* map PC keys in the array that simulates the
          Spectrum keyboard (key up)
         */
        switch(wParam)
        {
        case '1': keybd_buff[3] &= 0xFE; break;
        case '2': keybd_buff[3] &= 0xFD; break;
        case '3': keybd_buff[3] &= 0xFB; break;
        case '4': keybd_buff[3] &= 0xF7; break;
        case '5': keybd_buff[3] &= 0xEF; break;
        case 'Q': keybd_buff[2] &= 0xFE; break;
        case 'W': keybd_buff[2] &= 0xFD; break;
        case 'E': keybd_buff[2] &= 0xFB; break;
        case 'R': keybd_buff[2] &= 0xF7; break;
        case 'T': keybd_buff[2] &= 0xEF; break;
        case 'A': keybd_buff[1] &= 0xFE; break;
        case 'S': keybd_buff[1] &= 0xFD; break;
        case 'D': keybd_buff[1] &= 0xFB; break;
        case 'F': keybd_buff[1] &= 0xF7; break;
        case 'G': keybd_buff[1] &= 0xEF; break;
        case VK_SHIFT:
             if(((lParam >> 16) & 0x7F) == 0x2A)
                keybd_buff[0] &= 0xFE; /* CAPS SHIFT */
             else
                keybd_buff[7] &= 0xFD; /* SYMBOL SHIFT */
             break;
        case 'Z': keybd_buff[0] &= 0xFD; break;
        case 'X': keybd_buff[0] &= 0xFB; break;
        case 'C': keybd_buff[0] &= 0xF7; break;
        case VK_DIVIDE:
                 keybd_buff[7] &= 0xFD;
        case 'V': keybd_buff[0] &= 0xEF; break;
        case '0': keybd_buff[4] &= 0xFE; break;
        case '9': keybd_buff[4] &= 0xFD; break;
        case '8': keybd_buff[4] &= 0xFB; break;
        case '7': keybd_buff[4] &= 0xF7; break;
        case '6': keybd_buff[4] &= 0xEF; break;
        case 'P': keybd_buff[5] &= 0xFE; break;
```

```
case 'O': keybd_buff[5] &= 0xFD; break;
case 'I': keybd_buff[5] &= 0xFB; break;
case 'U': keybd_buff[5] &= 0xF7; break;
case 'Y': keybd_buff[5] &= 0xEF; break;
case VK_RETURN:keybd_buff[6] &= 0xFE; break;
case 'L': keybd_buff[6] &= 0xFD; break;
case VK_ADD:
        keybd_buff[7] &= 0xFD;
case 'K': keybd_buff[6] &= 0xFB; break;
case VK_SUBTRACT:
        keybd_buff[7] &= 0xFD;
case 'J': keybd_buff[6] &= 0xF7; break;
case 'H': keybd_buff[6] &= 0xEF; break;

case VK_ESCAPE:
        keybd_buff[0] &= 0xFE; /* CAPS SHIFT */

case VK_SPACE:      keybd_buff[7] &= 0xFE; break;
case 'M': keybd_buff[7] &= 0xFB; break;
case 'N': keybd_buff[7] &= 0xF7; break;
case VK_MULTIPLY:
        keybd_buff[7] &= 0xFD;
case 'B': keybd_buff[7] &= 0xEF; break;
case VK_TAB:
     keybd_buff[0] &= 0xFE;
     keybd_buff[7] &= 0xFD;
     break;

case VK_BACK: keybd_buff[0] &= 0xFE; /* CAPS SHIFT */
           keybd_buff[4] &= 0xFE;
           break;

/* kempston joystick */
case VK_LEFT:  joystick &= ~2; break;
case VK_RIGHT: joystick &= ~1; break;
case VK_UP:    joystick &= ~8; break;
case VK_DOWN:  joystick &= ~4; break;
case VK_CONTROL: joystick &= ~16; break;

 /* Sinclair joystick */
  case VK_NUMPAD5:
case VK_NUMPAD0: keybd_buff[0] &= 0xFE;
              keybd_buff[4] &= 0xFE; /* 0 - fire  */
              break;
case VK_NUMPAD4: keybd_buff[0] &= 0xFE;
              keybd_buff[3] &= 0xEF; /* 5 - left  */
              break;
case VK_NUMPAD6: keybd_buff[0] &= 0xFE;
              keybd_buff[4] &= 0xFB; /* 8 - right */
              break;
case VK_NUMPAD8: keybd_buff[0] &= 0xFE;
              keybd_buff[4] &= 0xF7; /* 7 - up    */
              break;
case VK_NUMPAD2: keybd_buff[0] &= 0xFE;
              keybd_buff[4] &= 0xEF; /* 6 - down  */
              break;

/* shortcut key for saveas and load a' la Z80 */

case VK_F2:
        PostMessage(hwnd, WM_COMMAND, IDM_SAVEAS, 0L);
        break;

case VK_F3:
        PostMessage(hwnd, WM_COMMAND, IDM_OPEN, 0L);
        break;

/* for PAUSE */
case VK_F4:
        PostMessage(hwnd, WM_COMMAND, IDM_PAUSE, 0L);
        break;
```

```
           /* for RESET */
           case VK_F5:
                   PostMessage(hwnd, WM_COMMAND, IDM_RESET, 0L);
                   break;

           }
           return 0L;

       case WM_DROPFILES:      // handles drag-and-drop messages (files)
          {
          HANDLE Handle = (HANDLE)wParam;
          int nFileLength;    // length of file name
          char *pszFileName;  // name of file

          nFileLength  = DragQueryFile( Handle , 0 , NULL, 0 );
          pszFileName = (char*)malloc(nFileLength + 1);
          DragQueryFile( Handle , 0, pszFileName, nFileLength + 1 );
          DragFinish( Handle );      // signal reception of message
          open_sna(pszFileName);     // open file
          free((char *)pszFileName); // free buffer

          // handle special case if paused
          if(!NotPaused)
            {
             NotPaused = 1;
             PostMessage(hwnd, WM_COMMAND, IDM_PAUSE, 0L);
            }

          return 0L;
          }

        case WM_CLOSE:                       // finish application
          DragAcceptFiles( hwnd , FALSE );  // close drag-and-drop
          /*KillTimer (hwnd, ID_TIMER) ;*/
#if defined(WINDOWS_SOUND)
          // Close wave driver
          CloseSnd();
#endif
          DestroyWindow(hwnd);
          PostQuitMessage(0);
          break;

        }

        // pass messages to windows default handler
        return DefWindowProc(hwnd,msg,wParam,lParam);
}


/*--------------------------------------------------------------------------*\
|   AppPaint(hwnd, hdc)                                                      |
|                                                                            |
|   Description:                                                             |
|       The paint function.                                                  |
|                                                                            |
|   Arguments:                                                               |
|       hwnd             window painting into                                |
|       hdc              display context to paint to                         |
|                                                                            |
|   Returns:                                                                 |
|       nothing                                                              |
|                                                                            |
\*--------------------------------------------------------------------------*/
BOOL AppPaint (HWND hwnd, HDC hdc)
{
   DWORD l;
   RECT rc;
#if defined (WIN32)
   BYTE *lpData;
#else
   BYTE __huge *lpData;
#endif
```

```
   GetClientRect (hwnd, &rc); // Get the size of the window on the screen

   // And slam the image onto the screen:
#if defined (WIN32)
   // ************************
   BitBlt(hdc, 0, 0, rc.right-rc.left, rc.bottom-rc.top, hdcImage, 0, 0, SRCCOPY);
   // ************************
#else
   WinGStretchBlt(hdc, 0, 0, rc.right-rc.left, rc.bottom-rc.top,
              hdcImage, 0, 0, X_CORD, Y_CORD);
#endif
   WindowDirty = 0;
   return TRUE;
}

/*-------------------------------------------------------------------------*\
|   pixel_host()                                                            |
|                                                                           |
|   Description:                                                            |
|       Draw a point of (x,y) coord, of colour [colour] in the WinG bitmap  |
|                                                                           |
\*-------------------------------------------------------------------------*/
void pixel_host(unsigned short x, unsigned short y, UCHAR colour)
{
  // The data in our bitmap image is in 8bit chunks. Each element is an
  // index value to the colortable that is associated with the bitmap,
  // which, since we took pains to create an 'identity palette', is also
  // the same as the palette currently selected into this device context.

#if defined (WIN32)
   *((BYTE *)image.data.lpIndex+((y*X_CORD)+x)) = colour+10;
#else
   *((BYTE __huge *)image.data.lpIndex+((y*X_CORD)+x)) = colour+10;
#endif
}


/*-------------------------------------------------------------------------*\
|   AppCommand(hwnd, msg, wParam, lParam )                                  |
|                                                                           |
|   Description:                                                            |
|       handles WM_COMMAND messages for the main window (hwndApp)           |
|       of the parent window's messages.                                    |
|                                                                           |
\*-------------------------------------------------------------------------*/
LONG AppCommand (HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam)
{
   FARPROC fpAbout;
   int idItem;
   //HWND hwndCtl;
   WORD wNotifyCode;

#if defined(WIN32)
   idItem = LOWORD(wParam);         // WIN32: control or menu item identifier
   hwndCtl = (HWND)lParam;          // WIN32: handle of control
   //wNotifyCode = HIWORD(wParam);    // WIN32: notification message
#else
   idItem = wParam;                 // WIN16: control or menu item identifier
   //hwndCtl = (HWND) LOWORD(lParam); // WIN16: handle of control
   //wNotifyCode = HIWORD(lParam);    // WIN16: notification message
#endif

   switch(idItem) {
      HMENU hmenu;
      BOOL  menu_state;
      static char CaptionCopy[100];

      case IDM_OPEN:
        open_menu(hwndApp);

        // handle special case if paused
```

```
  if(!NotPaused)
    {
      NotPaused = 1;
      PostMessage(hwnd, WM_COMMAND, IDM_PAUSE, 0L);
  }

  InvalidateRect(hwnd,NULL,FALSE);
  break;

case IDM_SAVE:
  save_snap();
  break;

case IDM_SAVEAS:
  open_menu_save(hwndApp);
  InvalidateRect(hwnd,NULL,FALSE);
  break;

case IDM_ABOUT:
  fpAbout = MakeProcInstance ((FARPROC)AppAbout, hInstApp);
  DialogBox(hInstApp, szAppName, hwnd, (DLGPROC)fpAbout);
  InvalidateRect(hwnd,NULL,FALSE);
  FreeProcInstance (fpAbout);
  break;

case IDM_FILERELOAD:
  reload_snap();
  break;

case IDM_EXIT:
  PostMessage(hwnd,WM_CLOSE,0,0L);
  break;


case IDM_PAUSE:
  {
     NotPaused ^= 1;
     if(!NotPaused)
     {
        // Just force a redraw
        InvalidateRect (hwndApp, NULL, FALSE);

        GetWindowText(hwnd, CaptionCopy, 99);
        SetWindowText(hwnd, "Spectrum Emulator [Paused]" );
     }
     else
        SetWindowText(hwnd, CaptionCopy);
  }
  break;

case IDM_POKE:
{
  FARPROC lpProcedure;
  HINSTANCE hInst;

  hInst=(HINSTANCE)GetWindowWord(hwnd,GWW_HINSTANCE);
  lpProcedure=MakeProcInstance((FARPROC)DoPoke,hInst);
  DialogBox(hInst,(LPCSTR)MAKEINTRESOURCE(SP_POKE),
            hwnd,(DLGPROC)lpProcedure);
  FreeProcInstance(lpProcedure);
  InvalidateRect(hwnd,NULL,FALSE);
  break;
}

case IDM_SPEED:
{
  FARPROC lpProcedure;
  HINSTANCE hInst;

  hInst=(HINSTANCE)GetWindowWord(hwnd,GWW_HINSTANCE);
  lpProcedure=MakeProcInstance((FARPROC)DoSpeed,hInst);
  DialogBox(hInst,(LPCSTR)MAKEINTRESOURCE(SP_SPEED),
```

```
                hwnd,(DLGPROC)lpProcedure);
      FreeProcInstance(lpProcedure);
      InvalidateRect(hwnd,NULL,FALSE);
      break;
    }

    case IDM_RESET:
      do_reset();
      if(NotPaused)
      {
         SetWindowText(hwnd, "Spectrum Emulator" );
      }
      else
         strcpy(CaptionCopy, "Spectrum Emulator");
      break;

    case IDM_NMI:
      do_nmi_int();
      break;

    case IDM_SOUND:
      bSoundOn ^= 1;
      hmenu = GetMenu(hwnd);
      CheckMenuItem(hmenu, IDM_SOUND, bSoundOn? MF_CHECKED :MF_UNCHECKED);
      break;

    case IDM_COLOUR:
      bFlashOn ^= 1;
      hmenu = GetMenu(hwnd);
      CheckMenuItem(hmenu, IDM_COLOUR,
         bFlashOn? MF_CHECKED : MF_UNCHECKED);
      break;

    case IDM_MODEL3:
      bModel3 ^= 1;
      hmenu = GetMenu(hwnd);
      CheckMenuItem(hmenu, IDM_MODEL3,
         bModel3? MF_CHECKED : MF_UNCHECKED);
      break;

    case IDM_SIZE1:
      Scale = 1;
      ResizeWindow();
      break;

    case IDM_SIZE2:
      Scale = 2;
      ResizeWindow();
      break;

    case IDM_SIZE3:
      Scale = 3;
      ResizeWindow();
      break;

    case IDM_SIZE4:
      Scale = 4;
      ResizeWindow();
      break;

    case IDM_SAVEOPTIONS:
      {
         char szBuf[260];

         GetWindowsDirectory((LPSTR)szBuf, 259);
         strcat(szBuf, "\\wspecem.ini");

         save_sna((LPSTR)szBuf);
      }
      break;
    }
  return 0L; // returning '0' means we processed this message.
```

```
}


/*--------------------------------------------------------------------------*\
|   AppAbout( hDlg, uiMessage, wParam, lParam )                              |
|                                                                            |
|   Description:                                                             |
|       This function handles messages belonging to the "About" dialog box.  |
|       The only message that it looks for is WM_COMMAND, indicating the use |
|       has pressed the "OK" button.  When this happens, it takes down       |
|       the dialog box.                                                      |
|                                                                            |
|   Arguments:                                                               |
|       hDlg            window handle of about dialog window                 |
|       uiMessage       message number                                       |
|       wParam          message-dependent                                    |
|       lParam          message-dependent                                    |
|                                                                            |
|   Returns:                                                                 |
|       TRUE if message has been processed, else FALSE                       |
|                                                                            |
\*--------------------------------------------------------------------------*/
BOOL FAR PASCAL AppAbout(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam)
{
   int idItem;
   //HWND hwndCtl;
   //WORD wNotifyCode;

   switch (msg) {
      case WM_INITDIALOG:
        // Here is where you could move the dialog around, change
        // some of the text that is going to be displayed, or other
        // things that you want to have happen right before the
        // dialog is brought up.
        return TRUE;

      case WM_COMMAND:
        // A 'command' has been recieved by the dialog. Very few are
        // actually possible in this dialog. Usually just the 'OK' button
#if defined(WIN32)
        idItem = LOWORD(wParam);         // WIN32: control or menu item identifier
        //hwndCtl = (HWND)lParam;          // WIN32: handle of control
        wNotifyCode = HIWORD(wParam);    // WIN32: notification message
#else
        idItem = wParam;                 // WIN16: control or menu item identifier
        //hwndCtl = (HWND) LOWORD(lParam); // WIN16: handle of control
        //wNotifyCode = HIWORD(lParam);    // WIN16: notification message
#endif
        if (idItem == IDOK) {
           EndDialog(hwnd,TRUE);
           }
        break;

        }
    return FALSE;
}



/*--------------------------------------------------------------------------*\
|   AppExit()                                                                |
|                                                                            |
|   Description:                                                             |
|       app is just about to exit, cleanup                                   |
|                                                                            |
\*--------------------------------------------------------------------------*/
void AppExit()
{
   // Clean up after ourselves...

   if (hdcImage) {
      // Remove our Device Context that we got from WinG:
```

```
      HBITMAP hbm;

      // Its not nice to delete a bitmap that is selected into a Device
      // Context, so lets swap in the original bitmap, which will return
      // to us our custom bitmap. You -did- remember to save the original
      // bitmap didn't you?
      hbm = (HBITMAP)SelectObject(hdcImage, gbmOldMonoBitmap);

      // Now we can delete the bitmap...
      DeleteObject(hbm);

      // ...and the Device Context
      DeleteDC(hdcImage);
      }

      if(hpalApp) {
        // And finally remove our Palette:
        DeleteObject(hpalApp);
        }
      Close_Z80Emu();
}



// Should be at snasave.c, but it needs to much windows specific info
int save_pcx(HFILE hfp)
{
   long i;
   UCHAR __huge * p;
   UCHAR byte;
   UCHAR f_time = 1;

   putbyte(10, hfp); /* Manufacturer == Paintbrush PCX */
   putbyte(5, hfp);  /* 3.0 with palette info          */
   putbyte(1, hfp);  /* .PCX run-length encoding       */

   putbyte(4, hfp);  /* bits per pixel */
   put2(0, hfp);     /* COORDS */
   put2(0, hfp);
   put2(X_CORD-1, hfp);
   put2(Y_CORD-1, hfp);
   put2(X_CORD, hfp); /* Horizontal resolution */
   put2(Y_CORD, hfp); /* Vertical resolution */

   /* Save Pallete as RGB, 16 colours */
   for(i = 0 ; i < 16 ; i++)
   {
      putbyte(rgbvals[i][0], hfp);
      putbyte(rgbvals[i][1], hfp);
      putbyte(rgbvals[i][2], hfp);
   }
   putbyte(0, hfp);
   putbyte(1, hfp); /* number of colour planes */
   put2(X_CORD/2, hfp); /* bytes per line */
   for(i = 0 ; i < 60 ; i++)
      putbyte(0, hfp);

   /* Now save image */
   p = (UCHAR __huge *)image.data.lpIndex;
   i = ((long)X_CORD * (long)Y_CORD)/2;

   while(i)
   {
      byte = ((*p++)-10 << 4);
      byte = byte | ((*p++)-10);
      if(byte >= 0xC0)
        putbyte(0xC1, hfp);
      putbyte(byte, hfp);
      i--;
   }
   return 0;
```

```
}

/* buggy function --- if it's corrected we'll save .BMP too
int save_dib(HFILE hfp)
{
   long i;

   UCHAR __huge * p;
   BITMAPFILEHEADER i_block;
   */
   // I suspected the problem is between this line
   /*
   i_block.bfType = 0x4d42;
   i_block.bfSize = (long)X_CORD * (long)Y_CORD+sizeof(BITMAPINFOHEADER)+
                 sizeof(RGBQUAD) * 256 + sizeof(BITMAPFILEHEADER);
   i_block.bfReserved1 = 0;
   i_block.bfReserved2 = 0;
   i_block.bfOffBits   = sizeof(BITMAPFILEHEADER) + sizeof(RGBQUAD)*256
                       + image.bi.biSize;
    */
   // and this one. Someone will give it a try?

   /*p = (UCHAR __huge *)&i_block;
   i=sizeof(BITMAPFILEHEADER);
   while(i--)
      putbyte(*p++, hfp);

   p = (UCHAR __huge *)&image.bi;
   i=sizeof(BITMAPINFOHEADER);
   while(i--)
      putbyte(*p++, hfp);

   p = (UCHAR __huge *)image.aColors;
   i=sizeof(RGBQUAD)*256;
   while(i --)
      putbyte(*p++, hfp);

   p = (UCHAR __huge *)image.data.lpIndex;
   i = (long)X_CORD * (long)Y_CORD;
   while(i--)
      putbyte(*p++, hfp);


   return 0;
}
*/
/* EOF: WSpecem.c */
```

1.2. kernel.c

This is where lit can be found the main Z80 execute() function:

```
/* Kernel.c: Z80 initialization and main cycle - basic support routines.
 *
 * Copyright 1996 Rui Fernando Ferreira Ribeiro.
 *
 */

/*
 * History:
 *     4th April 96:
 *                  . Modified R register handling -- about 10% speed
 *           increase -- see also [ld8bits.c]
 *                     . Modified central loop of emulation (execute()) for
 *              a faster emulation
 *                        . Spoted an error in Z80 emulation - handling IX or IY
 *           prefixes, if next instruction is not a HL instruction,
 *              the prefix will affect all instructions until if founds
 *              a HL instruction or a ED prefix -- but it's surprising
 *              how failed only 4 of nearly 2000 spectrum programs
```

```
 *              tested.
 *
 *     5th April 96:
 *                  . Modified handling of IX and IY prefixes --
 *            extensive changes in the code. This will fix the error
 *            spoted and will increase once more the emulation speed.
 */

#include "env.h"
#include "ivars.h"

/* Increment the lower 7 bits of R in each M1 cycle
*/
#define inc_R() (R++)

/* Opcode being interpreted - in IX or IY */
static UCHAR opcode;

/*=========================================================================*
 *                              do_reset                                   *
 *=========================================================================*/
void do_reset()
{
   /* CPU internal flags  */
   _IM = IFF1 = IFF2 =

       /* CPU registers */
   R_BIT7 = R = I =

   HL = BC = DE = AF = IX = IY = SP =

   /* alternative registers */
   HL2 = BC2 = DE2 = AF2 =

   /* flags CPU */
   flags._S = flags._Z = flags._X = flags._H = flags._Y =
   flags._P = flags._N = flags._C = 0;
   /* Interrupt counter */
   ResetTickCounter();
   /* Program Counter */
   PutPC(0);
}

/*=========================================================================*
 *                              execute                                    *
 *=========================================================================*/
void execute()
{
  /* Z80 main cycle */
  /* --> 0xDD e 0xFD are only 'switches' wich map IX or IY in HL
      till instruction end [but not in instructions prefixed by ED]
     --> 0xED, 0xCB are 'gates' to another sets of instructions
   */
  if(DelayEmVal)
     {

     while(clock_ticks < INT_TIME)
       {
       inc_R();
       /* Call funtion indexed by opcode */
       (*instruc_tabl[Getnextbyte()])();
       }

       // Spend a bit of time ---
       {
       USHORT i;

       i = DelayEmVal;
       while(i)
       {
          (void)do_nothing(&i);
          {
```

```
          USHORT j = i;

          while(j)
             (void)do_nothing(&j);
           }
       }
         }
     }
  else
     {
     while(clock_ticks < INT_TIME)
       {
       inc_R();
       /* Call funtion indexed by opcode */
       (*instruc_tabl[Getnextbyte()])();
       }
     }
  /* do_int_tasks(); */
  /* if interrupts activated */
  if(IFF1)
     {
     do_interrupt();
     }
  else
     ResetTickCounter();
}

/*=========================================================================*
 *                              execute_IX                                 *
 *=========================================================================*/
static void execute_IX()
{
   inc_R();  /* It appears to be this way */

   (*instruc_tablIX[Getnextbyte()])();
}

/*=========================================================================*
 *                              execute_IY                                 *
 *=========================================================================*/
static void execute_IY()
{
   inc_R();  /* It appears to be this way */

   /* Call function acording to opcode */
   (*instruc_tablIY[Getnextbyte()])();
}

/*=========================================================================*
 *                              execute_CB                                 *
 *=========================================================================*/
static void execute_CB()
{
   inc_R();
   (*instruc_tablCB[Getnextbyte()])();
}

/*=========================================================================*
 *                            execute_IXCB                                 *
 *=========================================================================*/
static void execute_IXCB()
{
   /*If IX or IY is active, then the next byte isnït
        a instruction, but a displacement for IX or IY
   */
   lastbyte = Getnextbyte();
   (*instruc_tablIXCB[Getnextbyte()])();
}

/*=========================================================================*
 *                              execute_CB                                 *
 *=========================================================================*/
```

```
static void execute_IYCB()
{
   /*If IX or IY is active, then the next byte isnït
        a instruction, but a displacement for IX or IY
   */
   lastbyte = Getnextbyte();
   (*instruc_tablIYCB[Getnextbyte()])();
}

/*=========================================================================*
 *                              execute_ED                                 *
 *=========================================================================*/
static void execute_ED()
{
   inc_R();
   (*instruc_tablED[Getnextbyte()])();
}

/* EOF: Kernel.c */
```

1.3. z80.h

This header file provides the modules with macros to access the virtual Z80.

```
/* Z80.h : Registers and primitives.
 *
 * Copyright 1996 Rui Fernando Ferreira Ribeiro.
 *
*/

#include "quirks.h"

/* Do not change the order!
*/
struct Z80word_reg
{
   USHORT bc, de, hl, af;
};

struct Z80byte_reg
{
#if !defined (M68000)
   UCHAR c, b, e, d, l, h, f, a;
#else
   UCHAR b, c, d, e, h, l, a, f;
#endif
};

union Z80Regs
{
   struct Z80word_reg x;
   struct Z80byte_reg k;
   USHORT wregs[4];
   UCHAR  bregs[8];
};


/*
 * Flags (F) :   7   6   5   4   3   2   1   0
 *             ---------------------------------
 *             | S | Z | X | H | Y | P | N | C |
 *             ---------------------------------
 */
struct CPU_flags
{
   USHORT  _S;   /* Sign                    */
   USHORT  _Z;   /* Zero                    */
```

```
   USHORT  _X;   /* Auxiliary Overflow  ?   */ /* Undocumented */
   USHORT  _H;   /* Half carry              */
   USHORT  _Y;   /* Auxliary Half carry ?   */ /* Undocumented */
   USHORT  _P;   /* Parity - overflow - P/V */
   USHORT  _N;   /* Subtraction             */
   USHORT  _C;   /* Carry                   */
};

/* Shortcuts */

#define BC   Z80Regs.x.bc     /* Z80Wreg(0) -- BC   */
#define DE   Z80Regs.x.de     /* Z80Wreg(1) -- DE   */
#define HL   Z80Regs.x.hl     /* Z80Wreg(2) -- HL   */
#define AF   Z80Regs.x.af     /* Z80Wreg(3) -- AF   */
#define B    Z80Regs.k.b      /* Z80Breg(0) -- B    */
#define C    Z80Regs.k.c      /* Z80Breg(1) -- C    */
#define D    Z80Regs.k.d      /* Z80Breg(2) -- D    */
#define E    Z80Regs.k.e      /* Z80Breg(3) -- E    */
#define H    Z80Regs.k.h      /* Z80Breg(4) -- H    */
#define L    Z80Regs.k.l      /* Z80Breg(5) -- L    */
#define A    Z80Regs.k.a      /* Z80Breg(6) -- A    */
#define F    Z80Regs.k.f      /* Z80Breg(7) -- F    */

/* Note: to use F or AF, first build_F() must be called
*/

#define Z80Wreg(elem) Z80Regs.wregs[elem]
#if !defined (M68000)
#define Z80Breg(elem) Z80Regs.bregs[(elem)^1]
#else
#define Z80Breg(elem) Z80Regs.bregs[elem]
#endif

/* Alternate register set */
/* In the 'real' Z80 there is no acess to the alternate register set as a bytes
  , but they might be needed to debugging purposes
*/
#define BC2  Z80Regs2.x.bc    /* Z80Wreg2(0) -- BC' */
#define DE2  Z80Regs2.x.de    /* Z80Wreg2(1) -- DE' */
#define HL2  Z80Regs2.x.hl    /* Z80Wreg2(2) -- HL' */
#define AF2  Z80Regs2.x.af    /* Z80Wreg2(3) -- AF' */
#define B2   Z80Regs2.k.b     /* Z80Breg2(0) -- B'  */
#define C2   Z80Regs2.k.c     /* Z80Breg2(1) -- C'  */
#define D2   Z80Regs2.k.d     /* Z80Breg2(2) -- D'  */
#define E2   Z80Regs2.k.e     /* Z80Breg2(3) -- E'  */
#define H2   Z80Regs2.k.h     /* Z80Breg2(4) -- H'  */
#define L2   Z80Regs2.k.l     /* Z80Breg2(5) -- L'  */
#define A2   Z80Regs2.k.a     /* Z80Breg2(6) -- A'  */
#define F2   Z80Regs2.k.f     /* Z80Breg2(7) -- F'  */

#define Z80Wreg2(elem) Z80Regs2.wregs[elem]
#if !defined (M68000)
#define Z80Breg2(elem) Z80Regs2.bregs[(elem)^1]
#else
#define Z80Breg2(elem) Z80Regs2.bregs[elem]
#endif

union Z80IX
{
   USHORT IX;
   UCHAR  bregs[2];
};

union Z80IY
{
   USHORT IY;
   UCHAR  bregs[2];
};

#define IX Z80IX.IX
#define IY Z80IY.IY
```

```
#if !defined (M68000)
#define HX Z80IX.bregs[1]
#define LX Z80IX.bregs[0]
#define HY Z80IY.bregs[1]
#define LY Z80IY.bregs[0]
#else
#define HX Z80IX.bregs[0]
#define LX Z80IX.bregs[1]
#define HY Z80IY.bregs[0]
#define LY Z80IY.bregs[1]
#endif

/* Clock ticks between each interrupt */
#define INT_TIME 69888L         /* Timing provided by Ian Collier */

/*
    Z80 Timings:

       Condition test       -->  1T
       Inc/Dec 8  bits reg  -->  1T
       Inc/Dec 16 bits reg  -->  2T
       PC+n                 -->  5T
       sum IX+n             -->  5T
       read a memory byte   -->  3T
       read a port   byte   -->  4T (3T + 1T in Wait)
       Nmi int              -->  6T (3T + 3T - push two words in stack)
       Int IM 0             --> 10T (3T + 3T + 4T from data bus read)
       Int IM 1             -->  6T (3T + 3T)
       Int IM 2             --> 16T (3T + 3T + 4T + 3T + 3T -- the last
                           6T being 2 memory acesses to read the pointer
                           built by reg I and the data bus value)
*/

/* EOF: Z80.h */
```

1.4. video.c

This file is fill the WinG buffer according to the Z80 writes in the Spectrum memory area.

```
/* Video.c: Translates hardware screen addresses to coord
 *     system, fill screen buffer and implements Flash.
 *
 * Copyright 1996 Rui Fernando Ferreira Ribeiro.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "env.h"


/* buffer caching the Spectrum attributes state */
static char attrib_z80[24][32];

//  Write a byte on Spectrum memory, at the attribute cache,
// and at the WinG buffer if needed
void writebyte(unsigned short adress, unsigned char byte)
{
   unsigned char i;
   UCHAR x, y;      /* coordenadas */
   static unsigned char colour = 0;  /* ultimo atributo   */
   static unsigned char ink = 0;     /* ink   (com flash) */
   static unsigned char paper = 0;   /* paper (com flash) */

   /* each memory acess = 3T */

   /* if ROM defined can't write in lower adresses
    */
   if(adress < 0x4000)
      return;
```

```
*(mem + adress) = byte;


if(adress < 0x5800) /* If adress lower than attributes adress */
{
   static ladress = 0x4000;
   static lbyte = 0;

   WindowDirty = 1;
   if((ladress != adress) || (adress == 0x4000))
   {
      /* put adress in univ x,y coords and read attribs corresponding with
       that coords i.e. x=0,y=0 ecran top
        */
      y = ((ladress >> 8) & 7) | ((ladress >> 2) & 0x38) |
            ((ladress >> 5) & 0xc0);
   /* if attrib different recalculates ink&paper */

   if(colour != attrib_z80[y>>3][x = (ladress & 0x1F)])
   {
      colour = attrib_z80[y>>3][x];
    paper = (colour >> 3) & 0xF;
    ink = (colour & 7) | ((colour >> 3) & 8);
   }

   x <<= 3;
   /* put in host 8 pixels corresponding to Spectrum pixels
    */
   for(i = 0 ; i < 8 ; i++)
   {
      pixel_host(x++, y, (lbyte & 0x80)?ink:paper);
    lbyte <<= 1;
   }
   ladress = adress;
   }
   lbyte = byte;
}
else
   if(adress < 0x5B00) /* If adress in attrib zone */
   {
      unsigned char k;

    WindowDirty = 1;
    /* if attrib changed */
    if(attrib_z80[y = (adress >> 5) & 0x1f][x = adress & 0x1f] != byte)
    {
       /* keep it
        */
       colour = attrib_z80[y][x] = byte;
       /* recalculate ink&paper
        */
       paper = (colour >> 3) & 0xF;
       ink = (colour & 7) | ((colour >> 3) & 8);

       /* Transform text coords in screen adress
        */
       adress = ((y & 7)<<5) | ((y & 0x18)<<8) | x | 0x4000;

       /* put ïtextï coords in graphic coords
        */
       y <<= 3;
       x <<= 3;

       /* Print corresponding attribut square (8 * 8 pixels)
        */
       for(k = 0 ; k < 8 ; k++)
       {
          byte = readbyte(adress);
          for(i = 0 ; i < 8 ; i++)
          {
             pixel_host(x++, y, (byte & 0x80)?ink:paper);
```

```
               byte <<= 1;
              }
              /* go to next scan
               */
              adress += 256;
              x -= 8;
              y++;
            }
          }
        }
}


// Do the real work for flash happening on the WinG buffer
void do_flash()
{
   UCHAR colour, x1, y1, x, y, k, paper, ink, byte, i;
   USHORT adress;

   // hack for flushing byte buffer
   writebyte(0x4000, readbyte(0x4000) );

   for(y1 = 0 ; y1 < 24 ; y1++)
   {
      for(x1 = 0 ; x1 < 32 ; x1++)
      {
        x = x1;
        y = y1;
        if((colour = attrib_z80[y][x]) & 0x80)
        {
           if(!FlashState)
            {
              paper = (colour >> 3) & 0xF;
              ink = (colour & 7) | ((colour >> 3) & 8);
           }
           else
           {
              ink   = (colour >> 3) & 0xF;
              paper = (colour & 7) | ((colour >> 3) & 8);
            }

           /* Transform text coords in screen adress
            */
           adress = ((y & 7)<<5) | ((y & 0x18)<<8) | x | 0x4000;

           /* put ïtextï coords in graphic coords
            */
           y <<= 3;
           x <<= 3;

           /* Print corresponding attribut square (8 * 8 pixels)
            */
           for(k = 0 ; k < 8 ; k++)
           {
              byte = readbyte(adress);
              for(i = 0 ; i < 8 ; i++)
              {
                 pixel_host(x++, y, (byte & 0x80)?ink:paper);
                byte <<= 1;
              }
              /* go to next scan
               */
              adress += 256;
              x -= 8;
              y++;
           }
         }
      }
   }
}

/* Video.c */
```

1.5. env.h

Finally, a file with helper definitions for the emulator.

```
/* Env.h : Z80 global environment.
 *
 * Copyright 1996 Rui Fernando Ferreira Ribeiro.
 */

/*
 *      History:
 *              . added RBIT_7 for R register improved handling
 *
 *      5th April 96:
 *              . modified for new IX/IY handling
 */

#include <windows.h>
#include "z80.h"
#include "iglobal.h"

#define BEGIN_RAM   16384
#define BEGIN_VIDEO BEGIN_RAM

/* Len of description of Z80 files */
#define LEN_DESCRIPT 30

/* For external declarations, see INIT.C */

/* vars to keep states of options for emulator */
extern unsigned char          bModel3;    /* Model 3 or Model 2   */
extern unsigned char        bFlashOn;   /* Flash on or off      */
extern unsigned char        bSoundOn;   /* Sound on or off      */
extern unsigned char        Scale;      /* Scale of main window */
extern unsigned char        ScreenUpdate;/* Screen update: 50 fps
                                    /ScreenUpdate
                                  */
extern unsigned short       DelayEmVal;  /* Configurable delay to slow down
                                     emulation
                                   */

/* vars needed to keep track of state of emulator */
extern unsigned char WindowDirty;      /* Have we to redraw window?         */
extern unsigned char FlashState;       /* Are colours inverted or normal?   */


/* Main registers        */
extern union Z80Regs  Z80Regs;

/* Alternative registers */
extern union Z80Regs  Z80Regs2;

extern union Z80IX Z80IX;
extern union Z80IY Z80IY;

extern USHORT PC;

extern USHORT  SP;
extern UCHAR   R, R_BIT7,  I;

extern struct CPU_flags  flags;

/* 'ticks' counter ; SPECTRUM Z80A - 3,5469MHz -
  70938 'ticks' between each int (50 per second)
*/
extern unsigned long  clock_ticks;

/* Interrupt mode - 0, 1 or 2 */
```

```
extern UCHAR _IM;

/* interrupt 'switch' interrupts and copy
 *
 *            IFF1  IFF2
 *  ----------------------
 *  | RESET  |  0  |  0  |
 *  |--------+-----+-----|
 *  | DI     |  0  |  0  |
 *  |--------+-----+-----|
 *  | EI     |  1  |  1  |
 *  |--------+-----+-----|
 *  | LD A,I |  .  |  .  | - P/V = IFF2
 *  |--------+-----+-----|
 *  | LD A,R |  .  |  .  | - P/V = IFF2
 *  |--------+-----+-----|
 *  | NMI    |  0  |  .  |
 *  |--------+-----+-----|
 *  | RETN   | IFF2|  .  |
 *  |--------+-----+-----|
 *  | INT    |  0  |  0  |
 *  |--------+-----+-----|
 *  | RETI   |  .  |  .  | - same as RET. (This instruction only exists
 *  ----------------------              to be recognized by external
 *                                      devices, so they know int is over)
 */
extern UCHAR IFF1, IFF2;

/* Used in DDCB or FDCB to keep shift of IX or IY
 */
extern UCHAR lastbyte;

/* 64k Mem
 */
extern UCHAR FAR * mem;

/* 'flag' WriteRom :
 *  0 : 16k of ROM, 48k of RAM
 *  1 : 64k of RAM
 */
extern UCHAR WriteRom;

/* 'flag' trace
 */
extern UCHAR TraceOn;

extern USHORT parity_table[256];

#define ResetTickCounter() clock_ticks = 0L;
#define int_time() (clock_ticks >= INT_TIME)
#define Force_interrupt() (clock_ticks = INT_TIME+1)

/* Increment counter by (cycles)
 */
#define T(cycles) (clock_ticks += cycles)

/* Decrement counter by (cycles)
 */
#define dT(cycles) (clock_ticks -= cycles)

/* Macro to put adress in PC
 */
#define PutPC(adress) PC = (USHORT)(adress)

/*  memory adressing - readbyte()
 */
#define readbyte(adress) (*(mem+(USHORT)(adress)))

/* read word pointed to PC and sum 2 to PC
 */
#define Getnextword() (PC += 2, readword((USHORT)(PC - 2)) )
```

```
/* read byte pointed to PC and increment it
 */
#define Getnextbyte() readbyte(PC++)

/* Convert unsigned char to signed short (with signal extension)
 */
#define ucharToUshort(uchar) ((signed short)((signed char)(uchar)))

#define pIX   (IX+ucharToUshort(Getnextbyte()))
#define pIY   (IY+ucharToUshort(Getnextbyte()))
#define pCBIX (IX+ucharToUshort(lastbyte))
#define pCBIY (IY+ucharToUshort(lastbyte))

/* build F from interpreter flags when needed
 */
#define build_F() F = (flags._C != 0) | ((flags._N != 0) << 1) | \
((flags._P != 0) << 2) | ((flags._Y != 0) << 3)| ((flags._H != 0) << 4) | \
((flags._X != 0) << 5) | ((flags._Z != 0) << 6) | ((flags._S != 0) << 7);

/* build interpreter flags from F
 */
#define read_F() flags._S=F & (UCHAR)BIT_7, flags._Z=F & (UCHAR)BIT_6, \
flags._X=F & (UCHAR)BIT_5, flags._H=F & (UCHAR)BIT_4, flags._Y=F & \
(UCHAR)BIT_3, flags._P=F & (UCHAR)BIT_2, flags._N = F & (UCHAR)BIT_1, \
flags._C =F & (UCHAR)BIT_0;




/* stack management - push and pop
 */
#define push(value) writeword((USHORT)(SP -=2), value)
#define pop() (SP += 2, (USHORT)readbyte(SP - 2) | \
            (((USHORT)readbyte(SP - 1)) << 8)  )

/* instead of a function, now we have a array
 */
#define   parity(x) (parity_table[x])

/* EOF: Env.h */
```

Index

R

S

T

U

V

W

Z