The SamRam

Source: file "TECHINFO.DOC" of Z80 emulator package. Comments by Yarek.

For our own Spectrums Johan Muizelaar and I built a piece of hardware we called the SamRam (which has nothing to do with the SAM Coupe, by the way!). It contains a monitor program and software to make snapshots of programs. It's still very useful and I still use it a lot. An explanation of its functions is to be found in chapter 3.

(...)

3. THE SAMRAM

3.1 Basic extensions

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

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

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

Select the SamRam by starting the emulator with the -s switch, or by selecting it from the F9 menu. Normal Basic functions as usual; the character set is different from the original one. There are four new commands: *RS, *MOVE, *SAVE and *SPECTRUM, and two new functions, DEC and HEX, which have replaced ASN and ACS. DEC takes a string argument containing a hexadecimal number, and returns the decimal value of it. HEX is the inverse of the DEC function, and yields a four-character string.

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

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

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

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

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

3.2 The NMI software

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

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

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

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

```
    H: Get the current ATTR colour from the screen at the cursor's
       current position, and store it in memory.  This colour will be
       used by the next command:
    Z: Put the colour on the screen
    G: Get a character from the screen
    P: Put the character on the screen
    R: Remove all screen data that is invisible by the ATTR colour
    L: Take a look at the bitmap below the ATTR colour codes
    T: Return to the main menu.  You can also return by pressing
       EDIT, or ESC in the emulator.
    B: Change border colour
    V: Clear the whole screen
```

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

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

```
    1: Scherm 1 opslaan        (Store screen 1 into buffer 2)
    2: Scherm 2 veranderen     (Edit screen 2)
    3: Schermen verwisselen    (Swap screens)
    4: Scherm 2 weghalen       (Remove screen 2)
```

These four functions are rather obvious, I believe.

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

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

Finally, the 'S' option. This option allows you to save a snapshot to tape or microdrive. I used it a lot on my real Spectrum, and it works just as well on the emulator. It is very useful is you want to load a .Z80 program back into a real Spectrum again. There are three 'switches' you can toggle. The active choice is indicated by a bright green box, inactive boxes are non-bright. You have to use EGA or VGA to be able to see it... The first switch lets you select whether the SamRam rom should be active if the program loads or not. This is only meaningful is you load it back in a SamRam again. Usually I want the SamRam rom to be active because I like the character set better. The second switch indicates whether the SamRam should save a 'loading screen', which it takes from screen buffer 2. If screen buffer 2 contains a screen, this switch will by default be on. Finally, the last switch lets you select the output media, tape or cartridge.

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

The Microdrive BASIC loader needs code in the SamRam rom to start the program (the RANDOMIZE USR 43 calls it). It won't be very difficult to write a standard BASIC loader that doesn't need this code, but I don't think many people desperately need it... Anyway, using the Multiface 128 you can write a compressed snapshot to cartridge which doesn't need the Multiface.

```
3.3  The built-in monitor


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

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

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

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

     The monitor commands:

         Q: Decrease the memory pointer by one.  You effectively shift one
            byte up.
         A: Increase the memory pointer, shifting one byte down.
         ENTER: Shift one instruction down: the memory pointer is
            increased by the length of first instruction displayed on
            screen.
         M: Change the value of the memory pointer.  For instance, M:M
            won't change it.
         P: Put.  The word operand supplied will be stored in the first MP
            memory, and the others will shift on place to the right.
            Usually, you'll want to store the memory pointer by P:M
         G: Get.  Typing G:1, G:2 or G:3 moves the value of one of the MP
            memories to the MP.
         B: Byte.  This command needs a byte operand; it will be poked
            into memory, and the memory pointer will move one up.
         I: Insert.  The same as B, except that you can poke more than one
            byte.  It continues to ask for bytes to poke until you type
            Enter on a blank line.
         #: Toggles the default number base between hexadecimal and
            decimal.
         F: Find.  You can enter up to ten bytes, which will be searched
            through memory.  Searching will stop at address 0, because
            since the search string is stored in shadow Ram, searching
            would otherwise not always terminate.  Typing Enter on a blank
            line starts the search.  Byte operands are entered as usual,
            but:
            - If a number bigger than 256 decimal is entered, it is
              treated as a word in the standard LSB/MSB format.  So, 1234
              will search for 34,12 hex in that order.  Note that 0012
              will search for 12, not 12,00.
            - A line starting with " decodes into the string of characters
              (up to ten) behind it.  Normally this would only be the
              first character.  So instead of typing "M "Y "N "A "M "E
              (space=enter here) you type "MYNAME.  Note that any
              terminating " will also be searched for!
            - An x is treated as a wildcard.  So if you search for CD x 80
              any call to a subroutine in the block 8000-80FF is a hit.
              If you search for x 8000, you'll see every one-byte
              instruction that has the address 8000 as operand.
         N: Continues the search started by F from the current MP.
         $: Displays one page of disassembly on screen.  In this mode,
            the following commands are possible:
            $: Back to the main screen
            7: [Shift 7 also works, cursor up]: Go to the previous page.
               The monitor stores the addresses of the previous eight
               pages only.
            Q: Go back one byte (decrease MP by one)
            A: Go one byte forward (increase MP by one)
            Z: Dump this screen to the printer, in ASCII format.  Redirect
               the RS232 output to a file, and run CONVERT on it to convert
               the CR's into CR/LF's before printing (or tell your printer
               to do the conversion).
            Every other key displays the next page of disassembly.
         K: List.  The same mode as with $ is entered, but instead of a
            disassembly the bytes with their ASCII characters are
            displayed.  Useful to look for text.
         C: Clear.  Fills blocks of memory with a specified value.  The
            monitor prompts with 'First', 'Last' and 'With'.  The 'Last'
            address is inclusive!
         D: Dump.  Prompts with 'First' and 'Last', and dumps a
            disassembly of the block between these addresses to the
            printer.  See remark at $-Z.  The 'Last' address is again
            inclusive.
         R: Registers.  If you press Enter after R, an overview of the
            registers contents is displayed.  If you type one of A,B,C,D,
            E,H,L,A',B',C',D',E',H',L',I,R,AF,BC,DE,HL,AF',BC',DE',HL',
            IX,IY,SP or PC, you can change the value of it.  Changing the
            value of SP also changes the PC and AF values by the way.  You
            cannot change the Interrupt mode or IFF.
         V: Verplaats.  (Move).  Prompts with 'From', 'To' and 'Length'.
            Obvious.
         S: Save.  Enter the start of the block you wish to save first.
            The monitor then prompts with 'Length'.  The block is saved
            without a header, as a normal data block (A, the flagbyte, is
            0FF)
         L: Load.  Loads a block of data from tape, at the specified
            address.  Normal data blocks, headers and blocks with non-
            standard flag bytes can be loaded.  The first byte in memory
            will contain the flag byte.  If the checksum isn't 0 after
            loading, indicating a tape error, you'll hear a beep.
         H: Header read.  Loads headers and displays the contents on
            screen.
```

(...)

5. TECHNICAL INFORMATION

5.6 The SamRam

The SamRam contains a 32K static CMOS Ram chip, *(two 16kB banks placed under the ZXROM - Yarek)* and some I/O logic for port 31. If this port is read, it returns the position of the joystick, as a normal Kempston joystickinterface would. If written to, the port controls a programmable latch chip (the 74LS259) which contains 8 latches:

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

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

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

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