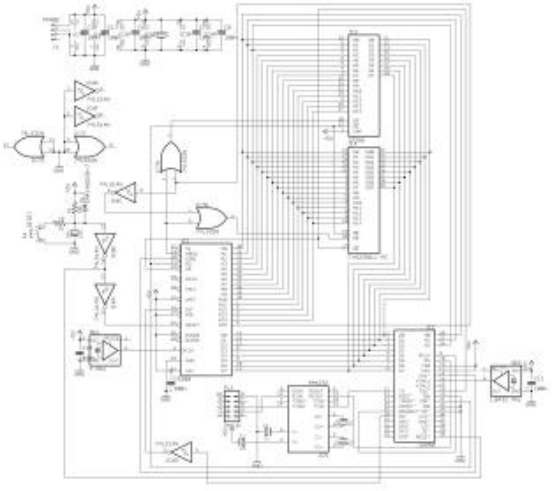
The Z80_mini

Quite some time ago, in October 2011, I built a simple Z80 based computer with an IDE interface, embedded Forth interpreter etc. This machine is described in more detail here. Since the overall system occupies a 10 inch enclosure and is completely wire-wrapped it is not as portable as I wished it was. Thus I decided to reimplement it and design a real printed circuit board using EAGLE - the free version of this CAD system allows one to design boards of up to 100 mm times 80 mm in size which is exactly the size I envisioned for this new Z80 system. This small size required some sacrifices - most notably there is longer an IDE-subsystem and the only IO-port is a serial line with which the Z80 board can be connected to a terminal or even better a host computer. This small computer is extremely simple to build, the printed circuit boards can be manufactured based on the EAGLE files enclosed below and using the builtin Forth interpreter is really fun.

The overall system is shown on the left. From left to right the bottom row of ICs contains a 16C550 UART (serial line interface), a Z80B processor running at a whopping 6 MHz, 32 kB of RAM (62256) and 32 kB EPROM (27C512) containing the monitor and the Forth interpreter. Above the RAM and ROM are the two TTL quartz oscillators for the UART (1.8432 MHz) and the CPU (6 MHz) and, on the far right, a 74LS32 for address decoding. The remaining parts are a MAX232 line driver for the serial line (next to the six small electrolytic capacitors) and a 74HCT14 which is mainly used for the reset circuitry (any 74xx14 should do fine here).

On the upper left, the power supply connector can be seen (a simple floppy disk like jack) with the tiny yellow reset switch just below. The ribbon cable connected on the upper right is the serial line.

The schematic of this board is shown on the right (click on the image to get a larger and readable version). In the center, the Z80 CPU can be seen. The reset circuit, which consists of an 10 uF electrolytic capacitor which is loaded via a 10k resistor and feeds two Schmitt trigger inverter gates of the 74HCT14, can be seen on the left of the drawing. When the reset switch is depressed, the capacitor is discharged and a suitable reset pulse is supplied to the CPU (active low) and the UART (active high).

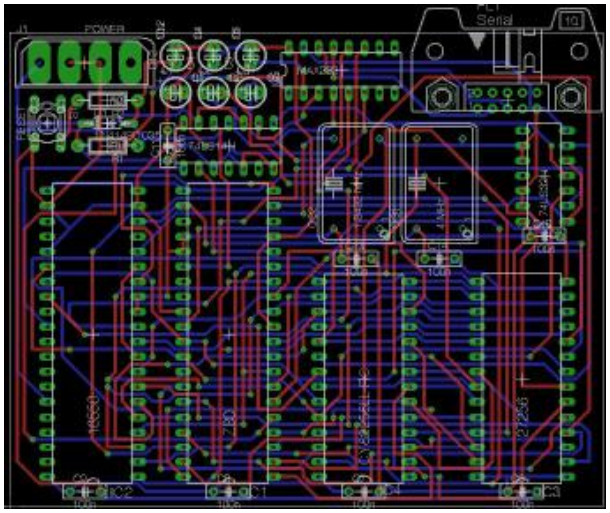


The 16 bit address space of the CPU is divided into two 32 kB areas by means of A15. The lower 32 kB from $0000 to $7FFF are occupied by ROM (27C512 EPROM) while the upper 32 kB from $8000 to $FFFF are RAM area.

The UART 16C550 with its associated level converter MAX232 can be seen in the lower right corner of the schematic.

Based on this schematic a printed circuit board was created using EAGLE's autorouter (as you can see I was lazy and did not take any special care of power supply lines since there are so few components which need no special precautions).

There are many companies which can create a board based on the Z80_mini.brd file which can be found in this ZIP-file. My



board was made by Jackaltac and I am very satisfied by the result. (No, I am not affiliated with this company nor do I intend to make any advertising - it is just that I have made some good experiences with their services.)

The overall system requires a 5 V power supply at about 120 mA with the parts specified above and running at 6 MHz.

A computer without software is useless, so I needed at least a simple EPROM based monitor for this computer to work with it at all. This monitor is based on the monitor of the predecessor Z80-system, the Tiny Z80 system mentioned above.

Writing a monitor turned out to be really fun and what once started with the goal of something capable of loading and running programs from Hex files became a monitor with embedded Forth interpreter (I ported Brad Rodriguez' CAMEL Forth) and many other useful features. (The monitor intended for the Tiny Z80 system contains routines for an IDE interface - these routines are also included in the monitor for the Z80_mini described here, but can not be used due to the missing hardware!)

I use the zasm assembler to assembly the monitor sources and produce an Intel-Hex file suitable for an EPROM programmer. The complete sources of the monitor can be found here. This ZIP-archive contains a README.txt file with additional information on how to build a monitor EPROM image file. If you just want to program a 27C512 EPROM with version 0.14a of the monitor (which is the latest version as of 28-JUN-2013) you can download a suitable Intel-Hex file here. (If you have no EPROM programmer you can send me a mail - see below - and ask me to program it for you given that you send me an empty 27(C)512 EPROM and the necessary postage and packaging to return it to you.)

On power on, the monitor prints this welcome-message:

```
Simple Z80-monitor - V 0.14a (B. Ulmann, September 2011 - June 2013)
   This monitor contains Brad Rodriguez' CAMEL Forth,
                         John Kerr's Z80 disassembler
--------------------------------------------------------------------------------
Z>
```

The command "language" is quite simple: All commands are grouped in so-called "command groups". Each such group is selected by a single letter and contains commands which are select by a second single letter. So there is no need to press "ENTER" as in conventional shells or type long commands. If a command requires parameters it prompts the user to enter all required values. There is one command group containing only one command, so that a single letter instead of two as written above is sufficient to execute this command which is the HELP-command:

```
Z> HELP: Known command groups and commands:

         C(ontrol group):
             C(old start), I(nfo), S(tart), W(arm start)
         D(isk group):
             I(nfo), M(ount), T(ransfer), U(nmount)
             R(ead), W(rite)
         F(ile group):
             C(at), D(irectory), L(oad), R(un)
         H(elp)
         M(emory group):
             (dis)A(ssemble), D(ump), E(xamine), F(ill), I(ntel Hex Load),
             L(oad), R(egister dump),
         S(ubsystem group):
             F(orth)
```

It should be noted again that all commands regarding disk and file IO will not work and will most likely crash the system since there is no IDE controller. I was too lazy to surgically remove all of the IDE support which is quite scattered within

the monitor. :-)

The following example shows the use of a command which requires a start and an end address to work - a disassembler listing covering the memory area from $0000 to $0050 is generated (the disassembler has been developed by John Kerr):

```
Z> MEMORY/DISASSEMBLE: START=0000 END=0050

0000 F3          DI
0001 31 3B FB    LD   SP,0FB3BH
0004 18 17       JR   1DH
0006 00          NOP
0007 00          NOP
0008 C5          PUSH BC
0009 E5          PUSH HL
000A DD E5       PUSH IX
000C E1          POP  HL
000D 29          ADD  HL,HL
000E 01 4C 1D    LD   BC,1D4CH
0011 09          ADD  HL,BC
0012 4E          LD   C,(HL)
0013 23          INC  HL
0014 46          LD   B,(HL)
0015 2B          DEC  HL
0016 C5          PUSH BC
0017 DD E1       POP  IX
0019 E1          POP  HL
001A C1          POP  BC
001B DD E9       JP   (IX)
001D 3E 80       LD   A,80H
001F D3 03       OUT  (3),A
0021 3E 0C       LD   A,0CH
0023 D3 00       OUT  (0),A
0025 AF          XOR  A
0026 D3 01       OUT  (1),A
0028 3E 03       LD   A,3
002A D3 03       OUT  (3),A
002C 3E 07       LD   A,7
002E D3 02       OUT  (2),A
0030 21 5A 01    LD   HL,15AH
0033 CD 4B 12    CALL 124BH
0036 3A 3C FB    LD   A,(0FB3CH)
0039 FE AA       CP   0AAH
003B 28 18       JR   Z,55H
003D 21 DA 02    LD   HL,2DAH
0040 CD 4B 12    CALL 124BH
0043 21 00 80    LD   HL,8000H
0046 54          LD   D,H
0047 5D          LD   E,L
0048 13          INC  DE
0049 01 FF 7F    LD   BC,7FFFH
004C 36 00       LD   (HL),0
004E ED B0       LDIR
0050 21 3C FB    LD   HL,0FB3CH

END OF DISASSEMBLER RUN.

Z>
```

To call the disassembler command only two keys have to be pressed: "M" to select the "M"emory-command-group and "A" to select the dis"A"ssembler.

Just in case you are as curious as I am and you like reading other people's code as much as I do, version 0.14a of the monitor is listed in the following (just in case Deft is reading this: There is plenty of room for improvement. :-) There are way too many push/pop instructions etc. But after all it was and still is a pet-project. :-)):

```
0000:           N8VEM           equ     0               ; If set to 1, the monitor is built
0000:                                                   ; for the N8VEM, otherwise, the
0000:                                                   ; old homebrew Z80 computer is targeted.
0000:           FEATURE_BASIC   equ     0               ; If set to 0, the BASIC interpreter
0000:                                                   ; will not be included.
0000:           ;==================================================================================
```

```
0000:           ;
0000:           ;  Small monitor for the Z80 single board computer consisting of 32 kB ROM
0000:           ; ($0000 to $ffff), 32 kB RAM ($8000 to $ffff) and a 16c550 UART.
0000:           ;
0000:           ; B. Ulmann, 28-SEP-2011, 29-SEP-2011, 01-OCT-2011, 02-OCT-2011, 30-OCT-2011,
0000:           ;            01-NOV-2011, 02-NOV-2011, 03-NOV-2011, 06/07/08-JAN-2012
0000:           ; I. Kloeckl, 06/07/08-JAN-2012 (FAT implementation for reading files)
0000:           ; B. Ulmann, 14-JAN-2011
0000:           ;
0000:           ; 07-MAR-2012: Fabio Zanicotti spotted an error in the ide_get_id-routine -
0000:           ;              the value $0a should have been written to ide_lba3 but was
0000:           ;              instead written into ide_status_cmd (I am quite puzzled why
0000:           ;              the routine worked nevertheless). This has been corrected. :-)
0000:           ;
0000:           ; B. Ulmann, 20-MAY-2012 port to N8VEM - no IDE support at the moment
0000:           ;            27-MAY-2012 PPIDE support for N8VEM
0000:           ;            29-MAY-2012 New system call uart_status introduced (used by the
0000:           ;                        4th interpreter)
0000:           ;            01-JUN-2012 CTRL-Y introduced
0000:           ;            03-JUN-2012 F/R command added, UART initialization changed
0000:           ;            06-JUN-2012 CAMEL-Forth added
0000:           ;            07-JUN-2012 Minor bug fixes (load routine), added rom2ram
0000:           ;            16-JUN-2012 stroup added, CAMEL Forth modified to use gets etc.
0000:           ;            12-MAY-2013 Added support for the old homebrew Z80 computer.
0000:           ;            20-MAY-2013 Added BASIC-subsystem.
0000:           ;            05-JUN-2013 Added John Kerr's Z80 disassembler.
0000:           ;            28-JUN-2013 Removed BASIC-support for the Z80mini
0000:           ;
0000:           ; Version 0.14a
0000:           ;
0000:           ;*****************************************************************************
0000:           ;
0000:           ; TODO:
0000:           ;       Read and print IDE error status codes in case of error!
0000:           ;
0000:           ; Known issues:
0000:           ;       Memory Dump has a problem when the end address is >= FF00
0000:           ;
0000:           ;*****************************************************************************
0000:           ;
0000:           ;  RST $00 will enter the monitor (do not care about the return address pushed
0000:           ; onto the stack - the stack pointer will be reinitialized during cold as well
0000:           ; as during warm starts.
0000:           ;
0000:           ;  Monitor routines will generally called by placing the necessary parameters
0000:           ; into some processor registers and then issuing RST $08. More about this later.
0000:           ;
0000:           ;  Whenever a call to the system routine getc is issued, it is tested if the
0000:           ; character entered was a CTRL-Y (like in VMS :-) ). If so, a restart of the
0000:           ; monitor takes place. Although this interrupt possibility requires a running
0000:           ; program to call getc from time to time it is better than nothing since the
0000:           ; monitor currently does not take care of interrupts at all.
0000:           ;
0000:           ;  Programs running in memory should not make use of memory above about $FB00
0000:           ; to leave some space for the monitor stack.
0000:           ;
0000:           ;  Memory layout is as follows:
0000:           ;
0000:           ;  +-------+
0000:           ;  ! $FFFF !    General purpose 512 byte buffer
0000:           ;  !  ---  !
0000:           ;  ! $FE00 !
0000:           ;  +-------+
0000:           ;  ! $DFFF !    FAT control block
0000:           ;  !  ---  !
0000:           ;  ! $FDDC !
0000:           ;  +-------+
0000:           ;  ! $FDDB !    File control block
0000:           ;  !  ---  !
0000:           ;  ! $FBBE !
0000:           ;  +-------+
0000:           ;  ! $FBBD !    81 byte string buffer
0000:           ;  !  ---  !
0000:           ;  ! $FB6D !
```

```
0000:           ;  +-------+
0000:           ;  ! $FB6C !    12 byte string buffer
0000:           ;  !  ---  !
0000:           ;  ! $FB61 !
0000:           ;  +-------+
0000:           ;  ! $FB60 !    Buffers for various routines
0000:           ;  !  ---  !
0000:           ;  ! $FB4D !
0000:           ;  +-------+
0000:           ;  ! $FB4C !    Scratch area (16 bytes)
0000:           ;  !  ---  !
0000:           ;  ! $FB3D !
0000:           ;  +-------+
0000:           ;  ! $FB3C !    Cold/warm start control (1 byte)
0000:           ;  +-------+
0000:           ;  ! $FB3B !    Stack
0000:           ;  !  ...  !
0000:           ;  ! $8000 !    Begin of RAM
0000:           ;  +-------+
0000:           ;  ! $7FFF !    ROM area
0000:           ;  !  ---  !    RST $08 calls a system routine
0000:           ;  ! $0000 !    RST $00 restarts the monitor
0000:           ;  +-------+
0000:           ;
0000:           ;
0000:           LOADABLE        equ     0               ; This is necessary for the built-in
0000:                                                   ; Forth-interpreter to work in ROM.
0000:           monitor_start   equ     $0000           ; $0000 -> ROM, $8000 -> Test image
0000:           #target rom             ; ((inserted by zasm))
0000:           #code   0,$10000        ; ((inserted by zasm))
0000:                           org     monitor_start
0000:           ;
0000:           ;  Definitions used by the monitor and all programs based on this monitor.
0000:           ;
0000:           ; 01-JUN-2013   B. Ulmann   1st implementation
0000:           ;
0000:           rom_start       equ     $0
0000:           rom_end         equ     $7fff
0000:           ram_start       equ     $8000
0000:           ram_end         equ     $ffff
0000:           buffer          equ     ram_end - $1ff  ; 512 byte IDE general purpose buffer
0000:           last_user_ram   equ     $f9ff
0000:           ;
0000:           ; Define the FAT control block memory addresses:
0000:           ;
0000:           datastart       equ     buffer - 4      ; Data area start vector
0000:           rootstart       equ     datastart - 4   ; Root directory start vector
0000:           fat1start       equ     rootstart - 4   ; Start vector to first FAT
0000:           psiz            equ     fat1start - 4   ; Size of partition (in sectors)
0000:           pstart          equ     psiz - 4        ; First sector of partition
0000:           rootlen         equ     pstart - 2      ; Maximum number of entries in directory
0000:           fatsec          equ     rootlen - 2     ; FAT size in sectors
0000:           ressec          equ     fatsec - 2      ; Number of reserved sectors
0000:           clusiz          equ     ressec - 1      ; Size of a cluster (in sectors)
0000:           fatname         equ     clusiz - 9      ; Name of the FAT (null terminated)
0000:           fatcb           equ     fatname         ; Start of the FATCB
0000:           ;
0000:           ; Define a file control block (FCB) memory addresses and displacements:
0000:           ;
0000:           file_buffer     equ     fatcb - $200            ; 512 byte sector buffer
0000:           cluster_sector  equ     file_buffer - 1         ; Current sector in cluster
0000:           current_sector  equ     cluster_sector - 4      ; Current sector address
0000:           current_cluster equ     current_sector - 2      ; Current cluster number
0000:           file_pointer    equ     current_cluster - 4     ; Pointer for file position
0000:           file_type       equ     file_pointer - 1        ; 0 -> not found, else OK
0000:           first_cluster   equ     file_type - 2           ; First cluster of file
0000:           file_size       equ     first_cluster - 4       ; Size of file
0000:           file_name       equ     file_size - 12          ; Canonical name of file
0000:           fcb             equ     file_name               ; Start of the FCB
0000:           ;
0000:           fcb_filename            equ     0
0000:           fcb_file_size           equ     $c
0000:           fcb_first_cluster       equ     $10
0000:           fcb_file_type           equ     $12
```

```
0000:           fcb_file_pointer        equ     $13
0000:           fcb_current_cluster     equ     $17
0000:           fcb_current_sector      equ     $19
0000:           fcb_cluster_sector      equ     $1d
0000:           fcb_file_buffer         equ     $1e
0000:           ;
0000:           ; We also need some general purpose string buffers:
0000:           ;
0000:           string_81_bfr   equ     fcb - 81
0000:           string_12_bfr   equ     string_81_bfr - 12
0000:           ;
0000:           ;  A number of routines need a bit of scratch RAM, too. Since these are
0000:           ; sometimes interdependent, each routine gets its own memory cells (only
0000:           ; possible since the routines are not recursively called).
0000:           ;
0000:           load_file_scrat equ     string_12_bfr - 2       ; Two bytes for load_file
0000:           str2filename_de equ     load_file_scrat - 2     ; Two bytes for str2filename
0000:           fopen_eob       equ     str2filename_de - 2     ; Eight bytes for fopen
0000:           fopen_rsc       equ     fopen_eob - 4
0000:           fopen_scr       equ     fopen_rsc - 2
0000:           dirlist_scratch equ     fopen_scr - 2           ; Eight bytes for fopen
0000:           dirlist_eob     equ     dirlist_scratch - 2
0000:           dirlist_rootsec equ     dirlist_eob - 4
0000:           scratch_area    equ     dirlist_rootsec - $1    ; Scratch memory (16 byte)
0000:           ;
0000:           start_type      equ     scratch_area  - $10    ; Distinguish cold/warm start
0000:           ;
0000:           ;  System calls are implemented by rst08 which expects the number of the
0000:           ; call to be executed in ix. The numbers of valid calls are defined here:
0000:           ;
0000:           _cold_start     equ     $0
0000:           _is_hex         equ     $1
0000:           _is_print       equ     $2
0000:           _to_upper       equ     $3
0000:           _crlf           equ     $4
0000:           _getc           equ     $5
0000:           _putc           equ     $6
0000:           _puts           equ     $7
0000:           _strcmp         equ     $8
0000:           _gets           equ     $9
0000:           _fgetc          equ     $a
0000:           _dump_fcb       equ     $b
0000:           _fopen          equ     $c
0000:           _dirlist        equ     $d
0000:           _fatmount       equ     $e
0000:           _fatunmount     equ     $f
0000:           _strchr         equ     $10
0000:           _uart_status    equ     $11
0000:           _getc_nowait    equ     $12
0000:           _print_word     equ     $13
0000:           _print_byte     equ     $14
0000:           _stroup         equ     $15
0000:           _get_word       equ     $16
0000:           ;
0000:           ; Some useful ASCII (control) characters:
0000:           ;
0000:           eos             equ     $00             ; End of string
0000:           cr              equ     $0d             ; Carriage return
0000:           lf              equ     $0a             ; Line feed
0000:           space           equ     $20             ; Space
0000:           tab             equ     $09             ; Tabulator
0000:           bs              equ     $08             ; Backspace
0000:           bel             equ     $07             ; Bell
0000:           ctrl_y          equ     25              ; CTRL-Y character
0000:
0000:           #include        "mondef.asm"
0000:           ;
0000:           #if N8VEM = 1
0000:           ;
0000:           ; 82C55 registers (the 82C55 is used to implement a simple IDE interface):
0000:           ;
0000:           reg_ppi_base    equ     $60                     ; 82C55 base address
0000:           reg_ppi_cntl    equ     reg_ppi_base + 3        ; 82C55 control register
0000:           ;
```

```
0000:           reg_ppide_lsb   equ     reg_ppi_base + 0        ; 82C55 port A
0000:           reg_ppide_msb   equ     reg_ppi_base + 1        ; 82C55 port B
0000:           reg_ppide_cntl  equ     reg_ppi_base + 2        ; 82C55 port C
0000:           #else
0000:           ide_base        equ     $10
0000:           #endif                                          ; N8VEM = 1?
0000:           ;
0000:           ; 16C550 registers:
0000:           ;
0000:           #if N8VEM = 1
0000:           uart_base       equ     $68
0000:           #else
0000:           uart_base       equ     $0
0000:           #endif
0000:           ;
0000:           uart_register_0 equ     uart_base + 0           ; Read/write a character
0000:           uart_register_1 equ     uart_base + 1
0000:           uart_register_2 equ     uart_base + 2
0000:           uart_register_3 equ     uart_base + 3
0000:           uart_register_4 equ     uart_base + 4
0000:           uart_register_5 equ     uart_base + 5           ; RX/TX status
0000:           uart_register_6 equ     uart_base + 6
0000:           uart_register_7 equ     uart_base + 7
0000:           ;
0000:           #if N8VEM = 1
0000:           ;
0000:           ; Memory Page Configuration Latches:
0000:           ;
0000:           mpcl_ram        equ     $78
0000:           mpcl_rom        equ     $7c
0000:           ;
0000:           #endif
0000:           ;
0000:           ; Main entry point (RST 00H):
0000:           ;
0000: F3        rst_00          di                      ; Disable interrupts
0001:           ;
0001:           ;  The stackpointer will be predecremented by a push instruction. Therefore
0001:           ; we set the start of the stack to the end of the reserved memory area in
0001:           ; high memory.
0001:           ;
0001: 313BFB                    ld      sp, start_type - $1
0004: 1817                      jr      initialize      ; Jump over the RST-area
0006:           ;
0006:           ;  RST-area - here is the main entry point into the monitor. The calling
0006:           ; standard looks like this:
0006:           ;
0006:           ; 1) Set register IX to the number of the system routine to be called.
0006:           ; 2) Set the remaining registers according to the routine's documentation.
0006:           ; 3) Execute RST $08 to actually call the system routine.
0006:           ; 4) Evaluate the values returned in the registers as described by the
0006:           ;    Routine's documentation.
0006:           ;
0006:           ;  (Currently there are no plans to use more RST entry points, so this routine
0006:           ; just runs as long as necessary in memory. If more RSTs will be used, this
0006:           ; routine should to be moved to the end of the used ROM area with only a
0006:           ; simple jump at the RST $08-location.)
0006:           ;
0006:           ;  This technique of calling system routines can be used as the following
0006:           ; example program that just echos characters read from the serial line
0006:           ; demonstrates:
0006:           ;
0006:           ;         org     $8000           ; Start in lower RAM
0006:           ; loop    ld      ix, 5           ; Prepare call to getc
0006:           ;         rst     08              ; Execute getc
0006:           ;         cp      3               ; CTRL-C pressed?
0006:           ;         jr      z, exit         ; Yes - exit
0006:           ;         ld      ix, 6           ; Prepare call to putc
0006:           ;         rst     08              ; Execute putx
0006:           ;         jr      loop            ; Process next character
0006:           ; exit    ld      ix, 4           ; Exit - print a CR/LF pair
0006:           ;         rst     08              ; Call CRLF
0006:           ;         ld      hl, msg         ; Pointer to exit message
0006:           ;         ld      ix, 7           ; Prepare calling puts
```

```
0006:           ;         rst     08              ; Call puts
0006:           ;         rst     00              ; Restart monitor (warm start)
0006:           ; msg     defb    "That's all folks.", $d, $a, 0
0006:           ;
0006:           ;  IMPORTANT: The content of ix is destroyed during the call, so it is NOT
0006:           ;             possible to perform successive calls to the same system service
0006:           ;             in a sequence without reloading the ix-register!
0006:           ;
0006:           ;  Currently the following functions are available (a more detailed description
0006:           ; can be found in the dispatch table itself - search for the label
0006:           ; dispatch_table):
0006:           ;
0006:           ;       $00:    cold_start
0006:           ;       $01:    is_hex
0006:           ;       $02:    is_print
0006:           ;       $03:    to_upper
0006:           ;       $04:    crlf
0006:           ;       $05:    getc
0006:           ;       $06:    putc
0006:           ;       $07:    puts
0006:           ;       $08:    strcmp
0006:           ;       $09:    gets
0006:           ;       $0A:    fgetc
0006:           ;       $0B:    dump_fcb
0006:           ;       $0C:    fopen
0006:           ;       $0D:    dirlist
0006:           ;       $0E:    fatmount
0006:           ;       $0F:    fatunmount
0006:           ;       $10:    strchr
0006:           ;       $11:    uart_status
0006:           ;       $12:    getc_nowait
0006:           ;       $13:    print_word
0006:           ;       $14:    print_byte
0006:           ;       $15:    stroup
0006:           ;       $16:    get_word
0006:           ;
0006:           ;               org     monitor_start + $08
0006: 00                        nop                     ; Beware: zasm is buggy concerning
0007: 00                        nop                     ; ORG. Therefore we need two nops to
0008:                                                   ; get to address $0008.
0008: C5        rst_08          push    bc              ; Save bc and hl
0009: E5                        push    hl
000A: DDE5                      push    ix              ; Copy the contents of ix
000C: E1                        pop     hl              ; into hl
000D: 29                        add     hl, hl          ; Double to get displacement in table
000E: 014C1D                    ld      bc, dispatch_table
0011: 09                        add     hl, bc          ; Calculate displacement in table
0012: 4E23462B                  ld      bc, (hl)        ; Load bc with the destination address
0016: C5                        push    bc
0017: DDE1                      pop     ix              ; Load ix with the destination address
0019: E1                        pop     hl              ; Restore hl
001A: C1                        pop     bc              ; and bc
001B: DDE9                      jp      (ix)            ; Jump to the destination routine
001D:           ;
001D:           ; Initialize UART to 9600,8N1:
001D:           ;
001D: 3E80      initialize      ld      a, $80
001F: D303                      out     (uart_register_3), a
0021: 3E0C                      ld      a, $c           ; 1843200 / (16 * 9600)
0023: D300                      out     (uart_register_0), a
0025: AF                        xor     a
0026: D301                      out     (uart_register_1), a
0028: 3E03                      ld      a, $3           ; 8N1
002A: D303                      out     (uart_register_3), a
002C: 3E07                      ld      a, $7           ; FIFO enable, reset RCVR/XMIT FIFO
002E: D302                      out     (uart_register_2), a
0030:           ;
0030:           ; Print welcome message:
0030:           ;
0030: 215A01                    ld      hl, hello_msg
0033: CD4B12                    call    puts
0036:           ;
0036:           ;  If this is a cold start (the location start_type does not contain $aa)
0036:           ; all available RAM will be reset to $00 and a message will be printed.
```

```
0036:           ;
0036: 3A3CFB    init_mem        ld      a, (start_type)
0039: FEAA                      cp      $aa             ; Warm start?
003B: 2818                      jr      z, main_loop    ; Yes - enter command loop
003D: 21DA02                    ld      hl, cold_start_msg
0040: CD4B12                    call    puts            ; Print cold start message
0043: 210080                    ld      hl, ram_start   ; Start of block to be filled with $00
0046: 545D                      ld      de, hl          ; End address of block
0048: 13                        inc     de              ; plus 1 (for ldir)
0049: 01FF7F                    ld      bc, ram_end - ram_start
004C: 3600                      ld      (hl), $00       ; Load first memory location
004E: EDB0                      ldir                    ; And copy this value down
0050: 213CFB                    ld      hl, start_type
0053: 36AA                      ld      (hl), $aa       ; Cold start done, remember this
0055:           ;
0055:           ; Read characters from the serial line and interpret them:
0055:           ;
0055: 216202    main_loop       ld      hl, monitor_prompt
0058: CD4B12                    call    puts
005B:           ;
005B:           ; The monitor is rather simple: All commands consist of one or two letters.
005B:           ; The first character selects a command group, the second the desired command
005B:           ; out of that group. When a command is recognized, it will be spelled out
005B:           ; automatically and the user will be prompted for arguments if applicable.
005B:           ;
005B: CDF902                    call    monitor_key     ; Read a key
005E:           ; Which group did we get?
005E: FE43                      cp      'C'             ; Control group?
0060: 2022                      jr      nz, disk_group  ; No - test next group
0062: 216802                    ld      hl, cg_msg      ; Print group prompt
0065: CD4B12                    call    puts
0068: CDF902                    call    monitor_key     ; Get command key
006B: FE43                      cp      'C'             ; Cold start?
006D: CA1C14                    jp      z, cold_start
0070: FE57                      cp      'W'             ; Warm start?
0072: CA2114                    jp      z, warm_start
0075: FE53                      cp      'S'             ; Start?
0077: CAC610                    jp      z, start
007A: FE49                      cp      'I'             ; Info?
007C: CC8B0D                    call    z, info
007F: 28D4                      jr      z, main_loop
0081: C34E01                    jp      cmd_error       ; Unknown control-group-command
0084: FE44      disk_group      cp      'D'             ; Disk group?
0086: 2028                      jr      nz, file_group  ; No - file group?
0088: 217102                    ld      hl, dg_msg      ; Print group prompt
008B: CD4B12                    call    puts
008E: CDF902                    call    monitor_key     ; Get command
0091: FE49                      cp      'I'             ; Info?
0093: CC1B08                    call    z, disk_info
0096: 28BD                      jr      z, main_loop
0098: FE4D                      cp      'M'             ; Mount?
009A: CC7C0F                    call    z, mount
009D: 28B6                      jr      z, main_loop
009F: FE54                      cp      'T'             ; Read from disk?
00A1: CC4708                    call    z, disk_transfer
00A4: 28AF                      jr      z, main_loop
00A6: FE55                      cp      'U'             ; Unmount?
00A8: CCE010                    call    z, unmount
00AB: 28A8                      jr      z, main_loop
00AD: C34E01                    jp      cmd_error       ; Unknown disk-group-command
00B0: FE46      file_group      cp      'F'             ; File group?
00B2: 2028                      jr      nz, help_group  ; No - help group?
00B4: 217702                    ld      hl, fg_msg      ; Print group prompt
00B7: CD4B12                    call    puts
00BA: CDF902                    call    monitor_key     ; Get command
00BD: FE43                      cp      'C'             ; Cat?
00BF: CC0A03                    call    z, cat_file
00C2: 2891                      jr      z, main_loop
00C4: FE44                      cp      'D'             ; Directory?
00C6: CC4903                    call    z, directory
00C9: 288A                      jr      z, main_loop
00CB: FE4C                      cp      'L'             ; Load?
00CD: CCDB0E                    call    z, load_file
00D0: 2883                      jr      z, main_loop
```

```
00D2: FE52                      cp      'R'             ; Run an executable?
00D4: CC6A0E                    call    z, load_and_run
00D7: CA5500                    jp      z, main_loop
00DA: 1872                      jr      cmd_error       ; Unknown file-group-command
00DC: FE48      help_group      cp      'H'             ; Help? (No further level expected.)
00DE: CCC40A                    call    z, help         ; Yes :-)
00E1: CA5500                    jp      z, main_loop
00E4: FE4D      memory_group    cp      'M'             ; Memory group?
00E6: C23401                    jp      nz, subsys_group; No - subsystem group?
00E9: 217D02                    ld      hl, mg_msg      ; Print group prompt
00EC: CD4B12                    call    puts
00EF: CDF902                    call    monitor_key     ; Get command key
00F2: FE41                      cp      'A'             ; Disassemble?
00F4: CC6103                    call    z, disassemble
00F7: CA5500                    jp      z, main_loop
00FA: FE44                      cp      'D'             ; Dump?
00FC: CC5409                    call    z, dump
00FF: CA5500                    jp      z, main_loop
0102: FE45                      cp      'E'             ; Examine?
0104: CCCA09                    call    z, examine
0107: CA5500                    jp      z, main_loop
010A: FE46                      cp      'F'             ; Fill?
010C: CC1D0A                    call    z, fill
010F: CA5500                    jp      z, main_loop
0112: FE49                      cp      'I'             ; INTEL-Hex load?
0114: CCD50C                    call    z, ih_load
0117: CA5500                    jp      z, main_loop
011A: FE4C                      cp      'L'             ; Load?
011C: CCA10D                    call    z, load
011F: CA5500                    jp      z, main_loop
0122: FE4D                      cp      'M'             ; Move?
0124: CC920F                    call    z, move
0127: CA5500                    jp      z, main_loop
012A: FE52                      cp      'R'             ; Register dump?
012C: CC0A10                    call    z, rdump
012F: CA5500                    jp      z, main_loop
0132:           #if N8VEM = 1
0132:                           cp      'S'             ; Switch ROM to RAM?
0132:                           call    z, rom2ram      ; This routine won't return
0132:                           jp      z, main_loop
0132:           #endif
0132: 181A                      jr      cmd_error       ; Unknown memory-group-command
0134: FE53      subsys_group    cp      'S'             ; Subsystem group?
0136: C24901                    jp      nz, group_error ; No - print an error message
0139: 218502                    ld      hl, sg_msg      ; Print group prompt
013C: CD4B12                    call    puts
013F: CDF902                    call    monitor_key     ; Get command key
0142: FE46                      cp      'F'             ; Forth?
0144: CA7A1D                    jp      z, forth_subsystem
0147:           #if FEATURE_BASIC = 1
0147:                           cp      'B'             ; BASIC?
0147:                           jp      z, basic_subsystem
0147:           #endif
0147: 1805                      jr      cmd_error       ; Unknown subsytem-group-command
0149: 21B602    group_error     ld      hl, group_err_msg
014C: 1803                      jr      print_error
014E: 219002    cmd_error       ld      hl, command_err_msg
0151: CD4012    print_error     call    putc            ; Echo the illegal character
0154: CD4B12                    call    puts            ; and print the error message
0157: C35500                    jp      main_loop
015A:           ;
015A:           ; Some constants for the monitor:
015A:           ;
015A: 0D0A0D0A
015E: 53696D70
0162: 6C65205A
0166: 38302D6D
016A: 6F6E6974
016E: 6F72202D
0172: 20562030
0176: 2E313461
017A: 20        hello_msg       defb    cr, lf, cr, lf, "Simple Z80-monitor - V 0.14a "
017B: 28422E20
017F: 556C6D61
```

```
0183: 6E6E2C20
0187: 53657074
018B: 656D6265
018F: 72203230
0193: 3131202D
0197: 204A756E
019B: 65203230
019F: 3133290D
01A3: 0A                        defb    "(B. Ulmann, September 2011 - June 2013)", cr, lf
01A4: 20202054
01A8: 68697320
01AC: 6D6F6E69
01B0: 746F7220
01B4: 636F6E74
01B8: 61696E73
01BC: 20427261
01C0: 6420526F
01C4: 64726967
01C8: 75657A27
01CC: 20                        defb    "   This monitor contains Brad Rodriguez' "
01CD: 43414D45
01D1: 4C20466F
01D5: 7274682C
01D9: 200D0A                    defb    "CAMEL Forth, ", cr, lf
01DC: 20202020
      ...
01F5: 4A6F686E
01F9: 204B6572
01FD: 72277320
0201: 5A383020                  defb    "                         John Kerr's Z80 "
0205: 64697361
0209: 7373656D
020D: 626C6572                  defb    "disassembler"
0211:           #if FEATURE_BASIC = 1
0211:                           defb    ", and", cr, lf
0211:                           defb    "                         BASIC 4.7 (C) Microsoft"
0211:           #endif
0211: 0D0A                      defb    cr, lf
0213: 2D2D2D2D
      ...
023B:                           defb    "----------------------------------------"
023B: 2D2D2D2D
      ...
0262:                           defb    "----------------------------------------"
0262: 0D0A5A3E
0266: 2000      monitor_prompt  defb    cr, lf, "Z> ", eos
0268: 434F4E54
026C: 524F4C2F
0270: 00        cg_msg          defb    "CONTROL/", eos
0271: 4449534B
0275: 2F00      dg_msg          defb    "DISK/", eos
0277: 46494C45
027B: 2F00      fg_msg          defb    "FILE/", eos
027D: 4D454D4F
0281: 52592F00  mg_msg          defb    "MEMORY/", eos
0285: 53554253
0289: 59535445
028D: 4D2F00    sg_msg          defb    "SUBSYSTEM/", eos
0290: 3A205379
0294: 6E746178
0298: 20657272
029C: 6F72202D
02A0: 20636F6D
02A4: 6D616E64
02A8: 206E6F74
02AC: 20666F75
02B0: 6E64210D
02B4: 0A00      command_err_msg defb    ": Syntax error - command not found!", cr, lf, eos
02B6: 3A205379
02BA: 6E746178
02BE: 20657272
02C2: 6F72202D
02C6: 2067726F
02CA: 7570206E
```

```
02CE: 6F742066
02D2: 6F756E64
02D6: 210D0A00  group_err_msg   defb    ": Syntax error - group not found!", cr, lf, eos
02DA: 436F6C64
02DE: 20737461
02E2: 72742C20
02E6: 636C6561
02EA: 72696E67
02EE: 206D656D
02F2: 6F72792E
02F6: 0D0A00    cold_start_msg  defb    "Cold start, clearing memory.", cr, lf, eos
02F9:           ;
02F9:           ;  Read a key for command group and command (this routine differs slightly
02F9:           ; from getc as it can also return to the monitor prompt with a dirty trick
02F9:           ; readjusting the stack):
02F9:           ;
02F9: CD5911    monitor_key     call    getc
02FC: FE0A                      cp      lf              ; Ignore LF
02FE: 28F9                      jr      z, monitor_key  ; Just get the next character
0300: CD4311                    call    to_upper
0303: FE0D                      cp      cr              ; A CR will return to the prompt
0305: C0                        ret     nz              ; No - just return
0306: 33                        inc     sp              ; Correct SP to and avoid ret!
0307: C35500                    jp      main_loop
030A:           ;
030A:           ;*****************************************************************************
030A:           ;***
030A:           ;*** The following routines are used in the interactive part of the monitor
030A:           ;***
030A:           ;*****************************************************************************
030A:           ;
030A:           ; Print a file's contents to STDOUT:
030A:           ;
030A: C5        cat_file        push    bc
030B: D5                        push    de
030C: E5                        push    hl
030D: FDE5                      push    iy
030F: 213A03                    ld      hl, cat_file_prompt
0312: CD4B12                    call    puts
0315: 216DFB                    ld      hl, string_81_bfr
0318: 0651                      ld      b, 81
031A: CDBA11                    call    gets            ; Read the filename into buffer
031D: CD5B12                    call    stroup          ; Convert to upper case
0320: FD21BEFB                  ld      iy, fcb         ; Prepare fopen (only one FCB currently)
0324: 1161FB                    ld      de, string_12_bfr
0327: CD1C17                    call    fopen
032A: CD6114    cat_file_loop   call    fgetc           ; Get a single character
032D: 3805                      jr      c, cat_file_exit
032F: CD4012                    call    putc            ; Print character if not EOF
0332: 18F6                      jr      cat_file_loop   ; Next character
0334: FDE1      cat_file_exit   pop     iy
0336: E1                        pop     hl
0337: D1                        pop     de
0338: C1                        pop     bc
0339: C9                        ret
033A: 4341543A
033E: 2046494C
0342: 454E414D
0346: 453D00    cat_file_prompt defb    "CAT: FILENAME=", eos
0349:           ;
0349:           ;  directory - a simple wrapper for dirlist (necessary for printing the command
0349:           ; name)
0349:           ;
0349: E5        directory       push    hl
034A: 215503                    ld      hl, directory_msg
034D: CD4B12                    call    puts
0350: CD6F18                    call    dirlist
0353: E1                        pop     hl
0354: C9                        ret
0355: 44495245
0359: 43544F52
035D: 590D0A00  directory_msg   defb    "DIRECTORY", cr, lf, eos
0361:           ;
0361:           ; Disassemble a memory area
```

```
0361:           ;
0361: F5        disassemble     push    af
0362: C5                        push    bc
0363: D5                        push    de
0364: E5                        push    hl
0365: DDE5                      push    ix
0367: 219D03                    ld      hl, dismsg1     ; Prompt for the start address
036A: CD4B12                    call    puts
036D: CDAF11                    call    get_word        ; Read user input
0370: E5                        push    hl              ; Save start address for later
0371: 21B103                    ld      hl, dismsg2     ; Prompt for end address
0374: CD4B12                    call    puts
0377: CDAF11                    call    get_word
037A: CD4C11                    call    crlf
037D: CD4C11                    call    crlf
0380: D1                        pop     de              ; Start address -> de
0381: E5                        push    hl              ; Push end address
0382: CDD403    disloop         call    disz80          ; Disassemble one instruction
0385: CD4C11                    call    crlf
0388: E1                        pop     hl
0389: E5                        push    hl
038A: A7                        and     a               ; Clear carry, just in case
038B: ED52                      sbc     hl, de          ; End address reached?
038D: 30F3                      jr      nc, disloop     ; No, continue disassembling
038F: 21B703                    ld      hl, dismsg3     ; Yes, print end message
0392: CD4B12                    call    puts
0395: E1                        pop     hl              ; Remove the end address copy
0396: DDE1                      pop     ix
0398: E1                        pop     hl
0399: D1                        pop     de
039A: C1                        pop     bc
039B: F1                        pop     af
039C: C9                        ret
039D: 44495341
03A1: 5353454D
03A5: 424C453A
03A9: 20535441
03AD: 52543D00  dismsg1 defb    "DISASSEMBLE: START=", eos
03B1: 20454E44
03B5: 3D00      dismsg2 defb    " END=", eos
03B7: 0D0A454E
03BB: 44204F46
03BF: 20444953
03C3: 41535345
03C7: 4D424C45
03CB: 52205255
03CF: 4E2E0D0A
03D3: 00        dismsg3 defb    cr, lf, "END OF DISASSEMBLER RUN.", cr, lf, eos
;... ...John Kerr's Disassembler resides here... ...
081A:           #include        "../disassembler/dis_z80.asm"
081B:           ;
081B:           ;
081B:           ; Get and print disk info:
081B:           ;
081B: F5        disk_info       push    af
081C: E5                        push    hl
081D: 213F08                    ld      hl, disk_info_msg
0820: CD4B12                    call    puts
0823: CDBC12                    call    ide_get_id      ; Read the disk info into the IDE buffer
0826: 2113FE                    ld      hl, buffer + $13
0829: 3609                      ld      (hl), tab
082B: CD4B12                    call    puts            ; Print vendor information
082E: CD4C11                    call    crlf
0831: 212DFE                    ld      hl, buffer + $2d
0834: 3609                      ld      (hl), tab
0836: CD4B12                    call    puts
0839: CD4C11                    call    crlf
083C: E1                        pop     hl
083D: F1                        pop     af
083E: C9                        ret
083F: 494E464F
0843: 3A0D0A00  disk_info_msg   defb    "INFO:", cr, lf, eos
0847:           ;
0847:           ; Read data from disk to memory
```

```
0847:           ;
0847: F5        disk_transfer   push    af
0848: C5                        push    bc
0849: D5                        push    de
084A: E5                        push    hl
084B: DDE5                      push    ix
084D: 21C608                    ld      hl, disk_trx_msg_0
0850: CD4B12                    call    puts            ; Print Read/Write prompt
0853: CD5911    disk_trx_rwlp   call    getc
0856: CD4311                    call    to_upper
0859: FE52                      cp      'R'             ; Read?
085B: 2009                      jr      nz, disk_trx_nr ; No
085D: DD215613                  ld      ix, ide_rs      ; Yes, we will call ide_rs later
0861: 21D008                    ld      hl, disk_trx_msg_1r
0864: 180B                      jr      disk_trx_main   ; Prompt the user for parameters
0866: FE57      disk_trx_nr     cp      'W'             ; Write?
0868: 20E9                      jr      nz, disk_trx_rwlp
086A: DD21C413                  ld      ix, ide_ws      ; Yes, we will call ide_ws later
086E: 21EA08                    ld      hl, disk_trx_msg_1w
0871: CD4B12    disk_trx_main   call    puts            ; Print start address prompt
0874: CDAF11                    call    get_word        ; Get memory start address
0877: E5                        push    hl
0878: 210509                    ld      hl, disk_trx_msg_2
087B: CD4B12                    call    puts            ; Prompt for number of blocks
087E: CD8A11                    call    get_byte        ; There are only 128 block of memory!
0881: FE00                      cp      0               ; Did the user ask for 00 blocks?
0883: 2008                      jr      nz, disk_trx_1  ; No, continue prompting
0885: 213209                    ld      hl, disk_trx_msg_4
0888: CD4B12                    call    puts
088B: 1830                      jr      disk_trx_exit
088D: 212309    disk_trx_1      ld      hl, disk_trx_msg_3
0890: CD4B12                    call    puts            ; Prompt for disk start sector
0893: CDAF11                    call    get_word        ; This is a four byte address!
0896: 444D                      ld      bc, hl
0898: CDAF11                    call    get_word
089B: 545D                      ld      de, hl
089D: E1                        pop     hl              ; Restore memory start address
089E:                           ; Register contents:
089E:                           ;       A:  Number of blocks
089E:                           ;       BC: LBA3/2
089E:                           ;       DE: LBA1/0
089E:                           ;       HL: Memory start address
089E: F5        disk_trx_loop   push    af              ; Save number of sectors
089F: CDC408                    call    disk_trampoline ; Read/write one sector (F is changed!)
08A2: E5                        push    hl              ; Save memory address
08A3: C5                        push    bc              ; Save LBA3/2
08A4: 626B                      ld      hl, de          ; Increment DE (LBA1/0)
08A6: 010100                    ld      bc, $0001       ; by one and
08A9: 09                        add     hl, bc          ; generate a carry if necessary
08AA: 545D                      ld      de, hl          ; Save new LBA1/0
08AC: E1                        pop     hl              ; Restore LBA3/2 into HL (!)
08AD: 3001                      jr      nc, disk_trx_skip
08AF: 09                        add     hl, bc          ; Increment BC if there was a carry
08B0: 444D      disk_trx_skip   ld      bc, hl          ; Write new LBA3/2 into BC
08B2: E1                        pop     hl              ; Restore memory address
08B3: C5                        push    bc              ; Save LBA3/2
08B4: 010002                    ld      bc, $200        ; 512 byte per block
08B7: 09                        add     hl, bc          ; Set pointer to next memory block
08B8: C1                        pop     bc              ; Restore LBA3/2
08B9: F1                        pop     af
08BA: 3D                        dec     a               ; One block already done
08BB: 20E1                      jr      nz, disk_trx_loop
08BD: DDE1      disk_trx_exit   pop     ix
08BF: E1                        pop     hl
08C0: D1                        pop     de
08C1: C1                        pop     bc
08C2: F1                        pop     af
08C3: C9                        ret
08C4: DDE9      disk_trampoline jp      (ix)
08C6: 5452414E
08CA: 53464552
08CE: 2F00      disk_trx_msg_0  defb    "TRANSFER/", eos
08D0: 52454144
08D4: 3A200D0A
```

```
08D8: 20202020
08DC: 4D454D4F
08E0: 52592053
08E4: 54415254
08E8: 3D00      disk_trx_msg_1r defb    "READ: ", cr, lf, "    MEMORY START=", eos
08EA: 57524954
08EE: 453A200D
08F2: 0A202020
08F6: 204D454D
08FA: 4F525920
08FE: 53544152
0902: 543D00    disk_trx_msg_1w defb    "WRITE: ", cr, lf, "    MEMORY START=", eos
0905: 204E554D
0909: 42455220
090D: 4F462042
0911: 4C4F434B
0915: 53202835
0919: 31322042
091D: 59544529
0921: 3D00      disk_trx_msg_2  defb    " NUMBER OF BLOCKS (512 BYTE)=", eos
0923: 20535441
0927: 52542053
092B: 4543544F
092F: 523D00    disk_trx_msg_3  defb    " START SECTOR=", eos
0932: 204E6F74
0936: 68696E67
093A: 20746F20
093E: 646F2066
0942: 6F72207A
0946: 65726F20
094A: 626C6F63
094E: 6B732E0D
0952: 0A00      disk_trx_msg_4  defb    " Nothing to do for zero blocks.", cr, lf, eos
0954:           ;
0954:           ; Dump a memory area
0954:           ;
0954: F5        dump            push    af
0955: C5                        push    bc
0956: D5                        push    de
0957: E5                        push    hl
0958: 21B409                    ld      hl, dump_msg_1
095B: CD4B12                    call    puts            ; Print prompt
095E: CDAF11                    call    get_word        ; Read start address
0961: E5                        push    hl              ; Save start address
0962: 21C109                    ld      hl, dump_msg_2  ; Prompt for end address
0965: CD4B12                    call    puts
0968: CDAF11                    call    get_word        ; Get end address
096B: CD4C11                    call    crlf
096E: 23                        inc     hl              ; Increment stop address for comparison
096F: 545D                      ld      de, hl          ; DE now contains the stop address
0971: E1                        pop     hl              ; HL is the start address again
0972:                           ; This loop will dump 16 memory locations at once - even
0972:                           ; if this turns out to be more than requested.
0972: 0610      dump_line       ld      b, $10          ; This loop will process 16 bytes
0974: E5                        push    hl              ; Save HL again
0975: CD3312                    call    print_word      ; Print address
0978: 21C709                    ld      hl, dump_msg_3  ; and a colon
097B: CD4B12                    call    puts
097E: E1                        pop hl                  ; Restore address
097F: E5                        push    hl              ; We will need HL for the ASCII dump
0980: 7E        dump_loop       ld      a, (hl)         ; Get the memory content
0981: CD1212                    call    print_byte      ; and print it
0984: 3E20                      ld      a, ' '          ; Print a space
0986: CD4012                    call    putc
0989: 23                        inc     hl              ; Increment address counter
098A: 10F4                      djnz    dump_loop       ; Continue with this line
098C:                           ; This loop will dump the very same 16 memory locations - but
098C:                           ; this time printable ASCII characters will be written.
098C: 0610                      ld      b, $10          ; 16 characters at a time
098E: 3E20                      ld      a, ' '          ; We need some spaces
0990: CD4012                    call    putc            ; to print
0993: CD4012                    call    putc
0996: E1                        pop     hl              ; Restore the start address
0997: 7E        dump_ascii_loop ld      a, (hl)         ; Get byte
```

```
0998: CD0A11                call    is_print        ; Is it printable?
099B: 3802                  jr      c, dump_al_1    ; Yes
099D: 3E2E                  ld      a, '.'          ; No - print a dot
099F: CD4012    dump_al_1   call    putc            ; Print the character
09A2: 23                    inc     hl              ; Increment address to read from
09A3: 10F2                  djnz    dump_ascii_loop
09A5:                       ; Now we are finished with printing one line of dump output.
09A5: CD4C11                call    crlf            ; CR/LF for next line on terminal
09A8: E5                    push    hl              ; Save the current address for later
09A9: A7                    and     a               ; Clear carry
09AA: ED52                  sbc     hl, de          ; Have we reached the last address?
09AC: E1                    pop     hl              ; restore the address
09AD: 38C3                  jr      c, dump_line    ; Dump next line of 16 bytes
09AF: E1                    pop     hl
09B0: D1                    pop     de
09B1: C1                    pop     bc
09B2: F1                    pop     af
09B3: C9                    ret
09B4: 44554D50
09B8: 3A205354
09BC: 4152543D
09C0: 00        dump_msg_1  defb    "DUMP: START=", eos
09C1: 20454E44
09C5: 3D00      dump_msg_2  defb    " END=", eos
09C7: 3A2000    dump_msg_3  defb    ": ", eos
09CA:           ;
09CA:           ; Examine a memory location:
09CA:           ;
09CA: F5        examine     push    af
09CB: E5                    push    hl
09CC: 21F809                ld      hl, examine_msg_1
09CF: CD4B12                call    puts
09D2: CDAF11                call    get_word        ; Wait for a four-nibble address
09D5: E5                    push    hl              ; Save address for later
09D6: 21160A                ld      hl, examine_msg_2
09D9: CD4B12                call    puts
09DC: E1        examine_loop pop    hl              ; Restore address
09DD: 7E                    ld      a, (hl)         ; Get content of address
09DE: 23                    inc     hl              ; Prepare for next examination
09DF: E5                    push    hl              ; Save hl again for later use
09E0: CD1212                call    print_byte      ; Print the byte
09E3: CD5911                call    getc            ; Get a character
09E6: FE20                  cp      ' '             ; A blank?
09E8: 2007                  jr      nz, examine_exit; No - exit
09EA: 3E20                  ld      a, ' '          ; Print a blank character
09EC: CD4012                call    putc
09EF: 18EB                  jr      examine_loop
09F1: E1        examine_exit pop    hl              ; Get rid of save hl value
09F2: CD4C11                call    crlf            ; Print CR/LF
09F5: E1                    pop     hl
09F6: F1                    pop     af
09F7: C9                    ret
09F8: 4558414D
09FC: 494E4520
0A00: 28747970
0A04: 65202720
0A08: 272F5245
0A0C: 54293A20
0A10: 41444452
0A14: 3D00      examine_msg_1 defb  "EXAMINE (type ' '/RET): ADDR=", eos
0A16: 20444154
0A1A: 413D00    examine_msg_2 defb  " DATA=", eos
0A1D:           ;
0A1D:           ; Fill a block of memory with a single byte - the user is prompted for the
0A1D:           ; start address, the length of the block and the fill value.
0A1D:           ;
0A1D: F5        fill        push    af              ; We will need nearly all registers
0A1E: C5                    push    bc
0A1F: D5                    push    de
0A20: E5                    push    hl
0A21: 21770A                ld      hl, fill_msg_1  ; Prompt for start address
0A24: CD4B12                call    puts
0A27: CDAF11                call    get_word        ; Get the start address
0A2A: E5                    push    hl              ; Store the start address
```

```
0A2B: A7                        and     a               ; Clear carry
0A2C: 010080                    ld      bc, ram_start
0A2F: ED42                      sbc     hl, bc          ; Is the address in the RAM area?
0A31: 3009                      jr      nc, fill_get_length
0A33: 21950A                    ld      hl, fill_msg_4  ; No!
0A36: CD4B12                    call    puts            ; Print error message
0A39: E1                        pop     hl              ; Clean up the stack
0A3A: 1836                      jr      fill_exit       ; Leave routine
0A3C: 21840A    fill_get_length ld      hl, fill_msg_2  ; Prompt for length information
0A3F: CD4B12                    call    puts
0A42: CDAF11                    call    get_word        ; Get the length of the block
0A45:                           ; Now make sure that start + length is still in RAM:
0A45: 444D                      ld      bc, hl          ; BC contains the length
0A47: E1                        pop     hl              ; HL now contains the start address
0A48: E5                        push    hl              ; Save the start address again
0A49: C5                        push    bc              ; Save the length
0A4A: 09                        add     hl, bc          ; Start + length
0A4B: A7                        and     a               ; Clear carry
0A4C: 010080                    ld      bc, ram_start
0A4F: ED42                      sbc     hl, bc          ; Compare with ram_start
0A51: 300A                      jr      nc, fill_get_value
0A53: 21A90A                    ld      hl, fill_msg_5  ; Print error message
0A56: CD4B12                    call    puts
0A59: C1                        pop     bc              ; Clean up the stack
0A5A: E1                        pop     hl
0A5B: 1815                      jr      fill_exit       ; Leave the routine
0A5D: 218D0A    fill_get_value  ld      hl, fill_msg_3  ; Prompt for fill value
0A60: CD4B12                    call    puts
0A63: CD8A11                    call    get_byte        ; Get the fill value
0A66: C1                        pop     bc              ; Get the length from the stack
0A67: E1                        pop     hl              ; Get the start address again
0A68: 545D                      ld      de, hl          ; DE = HL + 1
0A6A: 13                        inc     de
0A6B: 0B                        dec     bc
0A6C:                           ; HL = start address
0A6C:                           ; DE = destination address = HL + 1
0A6C:                           ;      Please note that this is necessary - LDIR does not
0A6C:                           ;      work with DE == HL. :-)
0A6C:                           ; A  = fill value
0A6C: 77                        ld      (hl), a         ; Store A into first memory location
0A6D: EDB0                      ldir                    ; Fill the memory
0A6F: CD4C11                    call    crlf
0A72: E1        fill_exit       pop     hl              ; Restore the register contents
0A73: D1                        pop     de
0A74: C1                        pop     bc
0A75: F1                        pop     af
0A76: C9                        ret
0A77: 46494C4C
0A7B: 3A205354
0A7F: 4152543D
0A83: 00        fill_msg_1      defb    "FILL: START=", eos
0A84: 204C454E
0A88: 4754483D
0A8C: 00        fill_msg_2      defb    " LENGTH=", eos
0A8D: 2056414C
0A91: 55453D00  fill_msg_3      defb    " VALUE=", eos
0A95: 20496C6C
0A99: 6567616C
0A9D: 20616464
0AA1: 72657373
0AA5: 210D0A00  fill_msg_4      defb    " Illegal address!", cr, lf, eos
0AA9: 20426C6F
0AAD: 636B2065
0AB1: 78636565
0AB5: 64732052
0AB9: 414D2061
0ABD: 72656121
0AC1: 0D0A00    fill_msg_5      defb    " Block exceeds RAM area!", cr, lf, eos
0AC4:           ;
0AC4:           ; Help
0AC4:           ;
0AC4: E5        help            push    hl
0AC5: 21CD0A                    ld      hl, help_msg
0AC8: CD4B12                    call    puts
```

```
0ACB: E1                    pop     hl
0ACC: C9                    ret
0ACD: 48454C50
0AD1: 3A204B6E
0AD5: 6F776E20
0AD9: 636F6D6D
0ADD: 616E6420
0AE1: 67726F75
0AE5: 70732061
0AE9: 6E642063
0AED: 6F6D6D61
0AF1: 6E64733A
0AF5: 0D0A      help_msg    defb    "HELP: Known command groups and commands:", cr, lf
0AF7: 0D0A                  defb    cr, lf
0AF9: 20202020
      ...
0B02: 43286F6E
0B06: 74726F6C
0B0A: 2067726F
0B0E: 7570293A
0B12: 0D0A                  defb    "         C(ontrol group):", cr, lf
0B14: 20202020
      ...
0B21: 43286F6C
0B25: 64207374
0B29: 61727429
0B2D: 2C204928
0B31: 6E666F29
0B35: 2C205328
0B39: 74617274
0B3D: 292C20                defb    "             C(old start), I(nfo), S(tart), "
0B40: 57286172
0B44: 6D207374
0B48: 61727429
0B4C: 0D0A                  defb    "W(arm start)", cr, lf
0B4E: 20202020
      ...
0B57: 44286973
0B5B: 6B206772
0B5F: 6F757029
0B63: 3A0D0A                defb    "         D(isk group):", cr, lf
0B66: 20202020
      ...
0B73: 49286E66
0B77: 6F292C20
0B7B: 4D286F75
0B7F: 6E74292C
0B83: 20542872
0B87: 616E7366
0B8B: 6572292C              defb    "             I(nfo), M(ount), T(ransfer),"
0B8F: 2055286E
0B93: 6D6F756E
0B97: 74290D0A              defb    " U(nmount)", cr, lf
0B9B: 20202020
      ...
0BA8: 52286561
0BAC: 64292C20
0BB0: 57287269
0BB4: 7465290D
0BB8: 0A                    defb    "             R(ead), W(rite)", cr, lf
0BB9: 20202020
      ...
0BC2: 4628696C
0BC6: 65206772
0BCA: 6F757029
0BCE: 3A0D0A                defb    "         F(ile group):", cr, lf
0BD1: 20202020
      ...
0BDE: 43286174
0BE2: 292C2044
0BE6: 28697265
0BEA: 63746F72
0BEE: 79292C20
0BF2: 4C286F61
```

```
0BF6: 64292C20
0BFA: 5228756E
0BFE: 290D0A                    defb    "             C(at), D(irectory), L(oad), R(un)", cr, lf
0C01: 20202020
      ...
0C0A: 4828656C
0C0E: 70290D0A                  defb    "         H(elp)", cr, lf
0C12: 20202020
      ...
0C1B: 4D28656D
0C1F: 6F727920
0C23: 67726F75
0C27: 70293A0D
0C2B: 0A                        defb    "         M(emory group):", cr, lf
0C2C: 20202020
      ...
0C39: 28646973
0C3D: 29412873
0C41: 73656D62
0C45: 6C65292C
0C49: 20442875
0C4D: 6D70292C
0C51: 20452878
0C55: 616D696E
0C59: 65292C20                  defb    "             (dis)A(ssemble), D(ump), E(xamine), "
0C5D: 4628696C
0C61: 6C292C20
0C65: 49286E74
0C69: 656C2048
0C6D: 6578204C
0C71: 6F616429
0C75: 2C200D0A                  defb    "F(ill), I(ntel Hex Load), ", cr, lf
0C79: 20202020
      ...
0C86: 4C286F61
0C8A: 64292C20
0C8E: 52286567
0C92: 69737465
0C96: 72206475
0C9A: 6D70292C
0C9E: 20                        defb    "             L(oad), R(egister dump), "
0C9F:           #if N8VEM = 1
0C9F:                           defb    "             S(witch ROM to RAM)", cr, lf
0C9F:           #else
0C9F: 0D0A                      defb    cr, lf
0CA1:           #endif
0CA1: 20202020
      ...
0CAA: 53287562
0CAE: 73797374
0CB2: 656D2067
0CB6: 726F7570
0CBA: 293A0D0A                  defb    "         S(ubsystem group):", cr, lf
0CBE: 20202020
      ...
0CCB: 46286F72
0CCF: 746829                    defb    "             F(orth)"
0CD2:           #if FEATURE_BASIC = 1
0CD2:                           defb    ", B(ASIC)"
0CD2:           #endif
0CD2: 0D0A00                    defb    cr, lf, eos
0CD5:           ;
0CD5:           ; Load an INTEL-Hex file (a ROM image) into memory. This routine has been
0CD5:           ; more or less stolen from a boot program written by Andrew Lynch and adapted
0CD5:           ; to this simple Z80 based machine.
0CD5:           ;
0CD5:           ; The INTEL-Hex format looks a bit awkward - a single line contains these
0CD5:           ; parts:
0CD5:           ; ':', Record length (2 hex characters), load address field (4 hex characters),
0CD5:           ; record type field (2 characters), data field (2 * n hex characters),
0CD5:           ; checksum field. Valid record types are 0 (data) and 1 (end of file).
0CD5:           ;
0CD5:           ; Please note that this routine will not echo what it read from stdin but
0CD5:           ; what it "understood". :-)
```

```
0CD5:           ;
0CD5: F5        ih_load         push    af
0CD6: D5                        push    de
0CD7: E5                        push    hl
0CD8: 21580D                    ld      hl, ih_load_msg_1
0CDB: CD4B12                    call    puts
0CDE: CD5911    ih_load_loop    call    getc            ; Get a single character
0CE1: FE0D                      cp      cr              ; Don't care about CR
0CE3: 28F9                      jr      z, ih_load_loop
0CE5: FE0A                      cp      lf              ; ...or LF
0CE7: 28F5                      jr      z, ih_load_loop
0CE9: FE20                      cp      space           ; ...or a space
0CEB: 28F1                      jr      z, ih_load_loop
0CED: CD4311                    call    to_upper        ; Convert to upper case
0CF0: CD4012                    call    putc            ; Echo character
0CF3: FE3A                      cp      ':'             ; Is it a colon?
0CF5: 204E                      jr      nz, ih_load_error
0CF7: CD8A11                    call    get_byte        ; Get record length into A
0CFA: 57                        ld      d, a            ; Length is now in D
0CFB: 1E00                      ld      e, $0           ; Clear checksum
0CFD: CD520D                    call    ih_load_chk     ; Compute checksum
0D00: CDAF11                    call    get_word        ; Get load address into HL
0D03: 7C                        ld      a, h            ; Update checksum by this address
0D04: CD520D                    call    ih_load_chk
0D07: 7D                        ld      a, l
0D08: CD520D                    call    ih_load_chk
0D0B: CD8A11                    call    get_byte        ; Get the record type
0D0E: CD520D                    call    ih_load_chk     ; Update checksum
0D11: FE01                      cp      $1              ; Have we reached the EOF marker?
0D13: 2012                      jr      nz, ih_load_data; No - get some data
0D15: CD8A11                    call    get_byte        ; Yes - EOF, read checksum data
0D18: CD520D                    call    ih_load_chk     ; Update our own checksum
0D1B: 7B                        ld      a, e
0D1C: A7                        and     a               ; Is our checksum zero (as expected)?
0D1D: 282C                      jr      z, ih_load_exit ; Yes - exit this routine
0D1F: 217A0D    ih_load_chk_err ld      hl, ih_load_msg_3
0D22: CD4B12                    call    puts            ; No - print an error message
0D25: 1824                      jr      ih_load_exit    ; and exit
0D27: 7A        ih_load_data    ld      a, d            ; Record length is now in A
0D28: A7                        and     a               ; Did we process all bytes?
0D29: 280B                      jr      z, ih_load_eol  ; Yes - process end of line
0D2B: CD8A11                    call    get_byte        ; Read two hex digits into A
0D2E: CD520D                    call    ih_load_chk     ; Update checksum
0D31: 77                        ld      (hl), a         ; Store byte into memory
0D32: 23                        inc     hl              ; Increment pointer
0D33: 15                        dec     d               ; Decrement remaining record length
0D34: 18F1                      jr      ih_load_data    ; Get next byte
0D36: CD8A11    ih_load_eol     call    get_byte        ; Read the last byte in the line
0D39: CD520D                    call    ih_load_chk     ; Update checksum
0D3C: 7B                        ld      a, e
0D3D: A7                        and     a               ; Is the checksum zero (as expected)?
0D3E: 20DF                      jr      nz, ih_load_chk_err
0D40: CD4C11                    call    crlf
0D43: 1899                      jr      ih_load_loop    ; Yes - read next line
0D45: 216B0D    ih_load_error   ld      hl, ih_load_msg_2
0D48: CD4B12                    call    puts            ; Print error message
0D4B: CD4C11    ih_load_exit    call    crlf
0D4E: E1                        pop     hl              ; Restore registers
0D4F: D1                        pop     de
0D50: F1                        pop     af
0D51: C9                        ret
0D52:           ;
0D52: 4F        ih_load_chk     ld      c, a            ; All in all compute E = E - A
0D53: 7B                        ld      a, e
0D54: 91                        sub     c
0D55: 5F                        ld      e, a
0D56: 79                        ld      a, c
0D57: C9                        ret
0D58: 494E5445
0D5C: 4C204845
0D60: 58204C4F
0D64: 41443A20
0D68: 0D0A00    ih_load_msg_1   defb    "INTEL HEX LOAD: ", cr, lf, eos
0D6B: 2053796E
```

```
0D6F: 74617820
0D73: 6572726F
0D77: 722100    ih_load_msg_2   defb    " Syntax error!", eos
0D7A: 20436865
0D7E: 636B7375
0D82: 6D206572
0D86: 726F7221
0D8A: 00        ih_load_msg_3   defb    " Checksum error!", eos
0D8B:           ;
0D8B:           ; Print version information etc.
0D8B:           ;
0D8B: E5        info            push    hl
0D8C: 219A0D                    ld      hl, info_msg
0D8F: CD4B12                    call    puts
0D92: 215A01                    ld      hl, hello_msg
0D95: CD4B12                    call    puts
0D98: E1                        pop     hl
0D99: C9                        ret
0D9A: 494E464F
0D9E: 3A2000    info_msg        defb    "INFO: ", eos
0DA1:           ;
0DA1:           ; Load data into memory. The user is prompted for a 16 bit start address. Then
0DA1:           ; a sequence of bytes in hexadecimal notation may be entered until a character
0DA1:           ; that is not 0-9 or a-f is encountered.
0DA1:           ;
0DA1: F5        load            push    af
0DA2: C5                        push    bc
0DA3: D5                        push    de
0DA4: E5                        push    hl
0DA5: 210C0E                    ld      hl, load_msg_1  ; Print command name
0DA8: CD4B12                    call    puts
0DAB: CDAF11                    call    get_word        ; Wait for the start address (2 bytes)
0DAE: E5                        push    hl              ; Remember address
0DAF: A7                        and     a               ; Clear carry
0DB0: 010080                    ld      bc, ram_start   ; Check if the address is valid
0DB3: ED42                      sbc     hl, bc          ; by subtracting the RAM start address
0DB5: E1                        pop     hl              ; Restore address
0DB6: 110000                    ld      de, 0           ; Counter for bytes loaded
0DB9: 3008                      jr      nc, load_loop   ; OK - start reading hex characters
0DBB: E5                        push    hl              ; Save address
0DBC: 213D0E                    ld      hl, load_msg_3  ; Print warning message
0DBF: CD4B12                    call    puts
0DC2: E1                        pop     hl              ; Restore address
0DC3:                           ;
0DC3:                           ; All in all we need two hex nibbles per byte. If two characters
0DC3:                           ; in a row are valid hexadecimal digits we will convert them
0DC3:                           ; to a byte and store this in memory. If one character is
0DC3:                           ; illegal, the load routine terminates and returns to the
0DC3:                           ; monitor.
0DC3:                           ;
0DC3: 3E20      load_loop       ld      a, ' '
0DC5: CD4012                    call    putc            ; Write a space as byte delimiter
0DC8: CD5911                    call    getc            ; Read first character
0DCB: CD4311                    call    to_upper        ; Convert to upper case
0DCE: CDF610                    call    is_hex          ; Is it a hex digit?
0DD1: 3026                      jr      nc, load_exit   ; No - exit the load routine
0DD3: CD1311                    call    nibble2val      ; Convert character to value
0DD6: CD2312                    call    print_nibble    ; Echo hex digit
0DD9: CB07                      rlc     a
0DDB: CB07                      rlc     a
0DDD: CB07                      rlc     a
0DDF: CB07                      rlc     a
0DE1: 47                        ld      b, a            ; Save the upper four bits for later
0DE2: CD5911                    call    getc            ; Read second character and proceed...
0DE5: CD4311                    call    to_upper        ; Convert to upper case
0DE8: CDF610                    call    is_hex
0DEB: 300C                      jr      nc, load_exit
0DED: CD1311                    call    nibble2val
0DF0: CD2312                    call    print_nibble
0DF3: B0                        or      b               ; Combine lower 4 bits with upper
0DF4: 77                        ld      (hl), a         ; Save value to memory
0DF5: 23                        inc     hl
0DF6: 13                        inc     de
0DF7: 18CA                      jr      load_loop       ; Get next byte (or at least try to)
```

```
0DF9: CD4C11    load_exit       call    crlf            ; Finished...
0DFC: 626B                      ld      hl, de          ; Print number of bytes loaded
0DFE: CD3312                    call    print_word
0E01: 212C0E                    ld      hl, load_msg_2
0E04: CD4B12                    call    puts
0E07: E1                        pop     hl
0E08: D1                        pop     de
0E09: C1                        pop     bc
0E0A: F1                        pop     af
0E0B: C9                        ret
0E0C: 4C4F4144
0E10: 20287878
0E14: 206F7220
0E18: 656C7365
0E1C: 20746F20
0E20: 656E6429
0E24: 3A204144
0E28: 44523D00  load_msg_1      defb    "LOAD (xx or else to end): ADDR=", eos
0E2C: 20627974
0E30: 6573206C
0E34: 6F616465
0E38: 642E0D0A
0E3C: 00        load_msg_2      defb    " bytes loaded.", cr, lf, eos
0E3D: 0D0A5772
0E41: 69746520
0E45: 746F206C
0E49: 6F776572
0E4D: 206D656D
0E51: 6F727920
0E55: 62616E6B
0E59: 2C207072
0E5D: 6F626162
0E61: 6C792052
0E65: 4F4D3A20  load_msg_3      defb    cr, lf, "Write to lower memory bank, probably ROM: "
0E69: 00                        defb    eos
0E6A:           ;
0E6A:           ;
0E6A:           ; Load and run an executable.
0E6A:           ;
0E6A: F5        load_and_run    push    af
0E6B: C5                        push    bc
0E6C: D5                        push    de
0E6D: E5                        push    hl
0E6E: FDE5                      push    iy
0E70: 21B20E                    ld      hl, lar_msg_1
0E73: CD4B12                    call    puts            ; Prompt for the filename
0E76: 216DFB                    ld      hl, string_81_bfr
0E79: 0651                      ld      b, 81           ; Buffer length
0E7B: CDBA11                    call    gets            ; Read filename
0E7E: CD5B12                    call    stroup          ; Convert to upper case
0E81: FD21BEFB                  ld      iy, fcb         ; Prepare file open operation
0E85: 1161FB                    ld      de, string_12_bfr
0E88: CD1C17                    call    fopen           ; Open the file
0E8B: 210080                    ld      hl, ram_start   ; The program is loaded into RAM
0E8E: 110000                    ld      de, 0           ; Counter for the number of bytes read
0E91: CD6114    lar_loop        call    fgetc           ; Get one byte from the file
0E94: 3805                      jr      c, lar_exit     ; EOF reached?
0E96: 77                        ld      (hl), a         ; Save one byte into RAM
0E97: 23                        inc     hl              ; Increment address pointer
0E98: 13                        inc     de              ; Increment counter of bytes loaded
0E99: 18F6                      jr      lar_loop        ; Get next byte
0E9B: CD4C11    lar_exit        call    crlf
0E9E: 626B                      ld      hl, de          ; How many bytes were read?
0EA0: CD3312                    call    print_word
0EA3: 21C10E                    ld      hl, lar_msg_2
0EA6: CD4B12                    call    puts
0EA9: FDE1                      pop     iy
0EAB: E1                        pop     hl
0EAC: D1                        pop     de
0EAD: C1                        pop     bc
0EAE: F1                        pop     af
0EAF: C30080                    jp      ram_start       ; Start the executable
0EB2: 52554E3A
0EB6: 2046494C
```

```
0EBA: 454E414D
0EBE: 453D00    lar_msg_1       defb    "RUN: FILENAME=", eos
0EC1: 20627974
0EC5: 6573206C
0EC9: 6F616465
0ECD: 642E2052
0ED1: 756E2E2E
0ED5: 2E0D0A0D
0ED9: 0A00      lar_msg_2       defb    " bytes loaded. Run...", cr, lf, cr, lf, eos
0EDB:           ;
0EDB:           ; Load a file's contents into memory:
0EDB:           ;
0EDB: F5        load_file       push    af
0EDC: C5                        push    bc
0EDD: D5                        push    de
0EDE: E5                        push    hl
0EDF: FDE5                      push    iy
0EE1: 213D0F                    ld      hl, load_file_msg_1
0EE4: CD4B12                    call    puts            ; Print first prompt (start address)
0EE7: CDAF11                    call    get_word        ; Wait for the start address (2 bytes)
0EEA: 225FFB                    ld      (load_file_scrat), hl
0EED: A7                        and     a               ; Clear carry
0EEE: 010080                    ld      bc, ram_start   ; Check if the address is valid
0EF1: ED42                      sbc     hl, bc          ; by subtracting the RAM start address
0EF3: 3008                      jr      nc, load_file_1
0EF5: 214E0F                    ld      hl, load_file_msg_2
0EF8: CD4B12                    call    puts
0EFB: 182B                      jr      load_file_exit  ; Illegal address - exit routine
0EFD: 21710F    load_file_1     ld      hl, load_file_msg_4
0F00: CD4B12                    call    puts            ; Prompt for filename
0F03: 216DFB                    ld      hl, string_81_bfr
0F06: 0651                      ld      b, 81           ; Buffer length
0F08: CDBA11                    call    gets            ; Read file name into bfr
0F0B: CD5B12                    call    stroup          ; Convert to upper case
0F0E: FD21BEFB                  ld      iy, fcb         ; Prepare open (only one FCB currently)
0F12: 1161FB                    ld      de, string_12_bfr
0F15: CD1C17                    call    fopen           ; Open the file (if possible)
0F18: 2A5FFB                    ld      hl, (load_file_scrat)
0F1B: 110000                    ld      de, 0           ; Counter for bytes loaded
0F1E: CD6114    load_file_loop  call    fgetc           ; Get one byte from the file
0F21: 3805                      jr      c, load_file_exit
0F23: 77                        ld      (hl), a         ; Store byte and
0F24: 23                        inc     hl              ; increment pointer
0F25: 13                        inc     de
0F26: 18F6                      jr      load_file_loop  ; Process next byte
0F28: CD4C11    load_file_exit  call    crlf
0F2B: 626B                      ld      hl, de          ; Print number of bytes loaded
0F2D: CD3312                    call    print_word
0F30: 21600F                    ld      hl, load_file_msg_3
0F33: CD4B12                    call    puts
0F36: FDE1                      pop     iy
0F38: E1                        pop     hl
0F39: D1                        pop     de
0F3A: C1                        pop     bc
0F3B: F1                        pop     af
0F3C: C9                        ret
0F3D: 4C4F4144
0F41: 2046494C
0F45: 453A2041
0F49: 4444523D
0F4D: 00        load_file_msg_1 defb    "LOAD FILE: ADDR=", eos
0F4E: 20496C6C
0F52: 6567616C
0F56: 20616464
0F5A: 72657373
0F5E: 2100      load_file_msg_2 defb    " Illegal address!", eos
0F60: 20627974
0F64: 6573206C
0F68: 6F616465
0F6C: 642E0D0A
0F70: 00        load_file_msg_3 defb    " bytes loaded.", cr, lf, eos
0F71: 2046494C
0F75: 454E414D
0F79: 453D00    load_file_msg_4 defb    " FILENAME=", eos
```

```
0F7C:           ;
0F7C:           ; mount - a wrapper for fatmount (necessary for printing the command's name)
0F7C:           ;
0F7C: E5        mount           push    hl
0F7D: 21880F                    ld      hl, mount_msg
0F80: CD4B12                    call    puts
0F83: CD4A1A                    call    fatmount
0F86: E1                        pop     hl
0F87: C9                        ret
0F88: 4D4F554E
0F8C: 540D0A0D
0F90: 0A00      mount_msg       defb    "MOUNT", cr, lf, cr, lf, eos
0F92:           ;
0F92:           ; Move a memory block - the user is prompted for all necessary data:
0F92:           ;
0F92: F5        move            push    af              ; We won't even destroy the flags!
0F93: C5                        push    bc
0F94: D5                        push    de
0F95: E5                        push    hl
0F96: 21D20F                    ld      hl, move_msg_1
0F99: CD4B12                    call    puts
0F9C: CDAF11                    call    get_word        ; Get address of block to be moved
0F9F: E5                        push    hl              ; Push this address
0FA0: 21DE0F                    ld      hl, move_msg_2
0FA3: CD4B12                    call    puts
0FA6: CDAF11                    call    get_word        ; Get destination start address
0FA9: 545D                      ld      de, hl          ; LDIR requires this in DE
0FAB:                           ; Is the destination address in RAM area?
0FAB: A7                        and     a               ; Clear carry
0FAC: 010080                    ld      bc, ram_start
0FAF: ED42                      sbc     hl, bc          ; Is the destination in RAM?
0FB1: 3009                      jr      nc, move_get_length
0FB3: 21EC0F                    ld      hl, move_msg_4  ; No - print error message
0FB6: CD4B12                    call    puts
0FB9: E1                        pop     hl              ; Clean up stack
0FBA: 180E                      jr      move_exit
0FBC: 21E30F    move_get_length ld      hl, move_msg_3
0FBF: CD4B12                    call    puts
0FC2: CDAF11                    call    get_word        ; Get length of block
0FC5: 444D                      ld      bc, hl          ; LDIR requires the length in BC
0FC7: E1                        pop     hl              ; Get address of block to be moved
0FC8:                           ; I was lazy - there is no test to make sure that the block
0FC8:                           ; to be moved will fit into the RAM area.
0FC8: EDB0                      ldir                    ; Move block
0FCA: CD4C11    move_exit       call    crlf            ; Finished
0FCD: E1                        pop     hl              ; Restore registers
0FCE: D1                        pop     de
0FCF: C1                        pop     bc
0FD0: F1                        pop     af
0FD1: C9                        ret
0FD2: 4D4F5645
0FD6: 3A204652
0FDA: 4F4D3D00  move_msg_1      defb    "MOVE: FROM=", eos
0FDE: 20544F3D
0FE2: 00        move_msg_2      defb    " TO=", eos
0FE3: 204C454E
0FE7: 4754483D
0FEB: 00        move_msg_3      defb    " LENGTH=", eos
0FEC: 20496C6C
0FF0: 6567616C
0FF4: 20646573
0FF8: 74696E61
0FFC: 74696F6E
1000: 20616464
1004: 72657373
1008: 2100      move_msg_4      defb    " Illegal destination address!", eos
100A:           ;
100A:           ; Dump the contents of both register banks:
100A:           ;
100A: F5        rdump           push    af
100B: E5                        push    hl
100C: 215110                    ld      hl, rdump_msg_1 ; Print first two lines
100F: CD4B12                    call    puts
1012: E1                        pop     hl
```

```
1013: CD8210                call    rdump_one_set
1016: D9                    exx
1017: 08                    ex      af, af'
1018: E5                    push    hl
1019: 216810                ld      hl, rdump_msg_2
101C: CD4B12                call    puts
101F: E1                    pop     hl
1020: CD8210                call    rdump_one_set
1023: 08                    ex      af, af'
1024: D9                    exx
1025: E5                    push    hl
1026: 216E10                ld      hl, rdump_msg_3
1029: CD4B12                call    puts
102C: DDE5                  push    ix
102E: E1                    pop     hl
102F: CD3312                call    print_word
1032: 217810                ld      hl, rdump_msg_4
1035: CD4B12                call    puts
1038: FDE5                  push    iy
103A: E1                    pop     hl
103B: CD3312                call    print_word
103E: 217D10                ld      hl, rdump_msg_5
1041: CD4B12                call    puts
1044: 210000                ld      hl, 0
1047: 39                    add     hl, sp
1048: CD3312                call    print_word
104B: CD4C11                call    crlf
104E: E1                    pop     hl
104F: F1                    pop     af
1050: C9                    ret
1051: 52454749
1055: 53544552
1059: 2044554D
105D: 500D0A0D
1061: 0A093173
1065: 743A00    rdump_msg_1     defb    "REGISTER DUMP", cr, lf, cr, lf, tab, "1st:", eos
1068: 09326E64
106C: 3A00      rdump_msg_2     defb    tab, "2nd:", eos
106E: 09505452
1072: 3A204958
1076: 3D00      rdump_msg_3     defb    tab, "PTR: IX=", eos
1078: 2049593D
107C: 00        rdump_msg_4     defb    " IY=", eos
107D: 2053503D
1081: 00        rdump_msg_5     defb    " SP=", eos
1082:           ;
1082: E5        rdump_one_set   push    hl              ; Print one register set
1083: 21B210                    ld      hl, rdump_os_msg_1
1086: CD4B12                    call    puts
1089: F5                        push    af              ; Move AF into HL
108A: E1                        pop     hl
108B: CD3312                    call    print_word      ; Print contents of AF
108E: 21B710                    ld      hl, rdump_os_msg_2
1091: CD4B12                    call    puts
1094: 6069                      ld      hl, bc
1096: CD3312                    call    print_word      ; Print contents of BC
1099: 21BC10                    ld      hl, rdump_os_msg_3
109C: CD4B12                    call    puts
109F: 626B                      ld      hl, de
10A1: CD3312                    call    print_word      ; Print contents of DE
10A4: 21C110                    ld      hl, rdump_os_msg_4
10A7: CD4B12                    call    puts
10AA: E1                        pop     hl              ; Restore original HL
10AB: CD3312                    call    print_word      ; Print contents of HL
10AE: CD4C11                    call    crlf
10B1: C9                        ret
10B2: 2041463D
10B6: 00        rdump_os_msg_1  defb    " AF=", eos
10B7: 2042433D
10BB: 00        rdump_os_msg_2  defb    " BC=", eos
10BC: 2044453D
10C0: 00        rdump_os_msg_3  defb    " DE=", eos
10C1: 20484C3D
10C5: 00        rdump_os_msg_4  defb    " HL=", eos
```

```
10C6:           ;
10C6:           #if N8VEM = 1
10C6:           ;
10C6:           ;  rom2ram copies the ROM contents to upper RAM, switches the lower 32 kB memory
10C6:           ; bank to RAM, and copies the ROM contents back. This functionality is quite
10C6:           ; handy to perform patches to the monitor without having to burn an EPROM.
10C6:           ;
10C6:           rom2ram         push    hl
10C6:                           ld      hl, rom2ram_m1
10C6:                           call    puts
10C6:                           ld      hl, rom_start   ; Copy from this address
10C6:                           ld      de, ram_start   ; to this address
10C6:                           ld      bc, end_of_monitor - rom_start + 1
10C6:                           ldir                    ; Copy BC bytes
10C6:                           ;  Now we have a copy of this monitor in the upper 32 KB
10C6:                           ; RAM bank. We will now jump into this copy to perform the
10C6:                           ; second copy step without crashing:
10C6:                           jp      rom2ram_switch + ram_start
10C6:           rom2ram_switch  ld      a, $8f          ; Select highest RAM bank
10C6:                           out     (mpcl_ram), a   ; for lower 32 kB of memory
10C6:                           ld      a, $80          ; Disable ROM, enable RAM
10C6:                           out     (mpcl_rom), a   ; Switch it!
10C6:                           ld      hl, ram_start   ; Prepare second copy step - source,
10C6:                           ld      de, rom_start   ; destination, length remains the same
10C6:                           ldir                    ; Copy block
10C6:                           ld      hl, rom2ram_m2  ; Print completion message
10C6:                           call    puts
10C6:                           pop     hl
10C6:                           ret
10C6:           rom2ram_m1      defb    "ROM2RAM", cr, lf, eos
10C6:           rom2ram_m2      defb    tab, "The monitor is now running in lower RAM."
10C6:                           defb    cr, lf, eos
10C6:           ;
10C6:           #endif
10C6:           ;
10C6:           ; Start a program - this will prompt for a four digital hexadecimal start
10C6:           ; address. A program should end with "jp $0" to enter the monitor again.
10C6:           ;
10C6: 21D310    start           ld      hl, start_msg
10C9: CD4B12                    call    puts
10CC: CDAF11                    call    get_word        ; Wait for a four-nibble address
10CF: CD4C11                    call    crlf
10D2: E9                        jp      (hl)            ; Start program (and hope for the best)
10D3: 53544152
10D7: 543A2041
10DB: 4444523D
10DF: 00        start_msg       defb    "START: ADDR=", eos
10E0:           ;
10E0:           ;
10E0:           ;  unmount - simple wrapper for fatunmount (necessary for printing the command
10E0:           ; name)
10E0:           ;
10E0: E5        unmount         push    hl
10E1: 21EC10                    ld      hl, unmount_msg
10E4: CD4B12                    call    puts
10E7: CD421D                    call    fatunmount
10EA: E1                        pop     hl
10EB: C9                        ret
10EC: 554E4D4F
10F0: 554E540D
10F4: 0A00      unmount_msg     defb    "UNMOUNT", cr, lf, eos
10F6:           ;
10F6:           ;*******************************************************************************
10F6:           ;***
10F6:           ;*** String routines
10F6:           ;***
10F6:           ;*******************************************************************************
10F6:           ;
10F6:           ; is_hex checks a character stored in A for being a valid hexadecimal digit.
10F6:           ; A valid hexadecimal digit is denoted by a set C flag.
10F6:           ;
10F6: FE47      is_hex          cp      'F' + 1         ; Greater than 'F'?
10F8: D0                        ret     nc              ; Yes
10F9: FE30                      cp      '0'             ; Less than '0'?
```

```
10FB: 3002                  jr      nc, is_hex_1    ; No, continue
10FD: 3F                    ccf                     ; Complement carry (i.e. clear it)
10FE: C9                    ret
10FF: FE3A      is_hex_1    cp      '9' + 1         ; Less or equal '9*?
1101: D8                    ret     c               ; Yes
1102: FE41                  cp      'A'             ; Less than 'A'?
1104: 3002                  jr      nc, is_hex_2    ; No, continue
1106: 3F                    ccf                     ; Yes - clear carry and return
1107: C9                    ret
1108: 37        is_hex_2    scf                     ; Set carry
1109: C9                    ret
110A:           ;
110A:           ; is_print checks if a character is a printable ASCII character. A valid
110A:           ; character is denoted by a set C flag.
110A:           ;
110A: FE20      is_print    cp      space
110C: 3002                  jr      nc, is_print_1
110E: 3F                    ccf
110F: C9                    ret
1110: FE7F      is_print_1  cp      $7f
1112: C9                    ret
1113:           ;
1113:           ; nibble2val expects a hexadecimal digit (upper case!) in A and returns the
1113:           ; corresponding value in A.
1113:           ;
1113: FE3A      nibble2val  cp      '9' + 1         ; Is it a digit (less or equal '9')?
1115: 3802                  jr      c, nibble2val_1 ; Yes
1117: D607                  sub     7               ; Adjust for A-F
1119: D630      nibble2val_1 sub    '0'             ; Fold back to 0..15
111B: E60F                  and     $f              ; Only return lower 4 bits
111D: C9                    ret
111E:           ;
111E:           ; strchr: HL points to the string to be searched, A contains the desired
111E:           ;         character. On return HL contains the address of the position where
111E:           ;         the character was found. If carry is set upon return the character
111E:           ;         was found.
111E:           ;
111E: C5        strchr      push    bc
111F: 47                    ld      b, a                ; Remember character
1120: 7E        strchr_loop ld      a, (hl)             ; Compare one character
1121: FE00                  cp      0                   ; Not really necessary...
1123: 2806                  jr      z, strchr_not       ; Terminating 0 found?
1125: B8                    cp      b                   ; Is it the one we are lkg. for?
1126: 2806                  jr      z, strchr_found     ; Yes!
1128: 23                    inc     hl                  ; Increment pointer
1129: 18F5                  jr      strchr_loop         ; Cmp. next character
112B: A7        strchr_not  and     a                   ; Reset carry flag
112C: 1801                  jr      strchr_exit
112E: 37        strchr_found scf                        ; Set carry
112F: C1        strchr_exit pop     bc
1130: C9                    ret
1131:           ;
1131:           ;  Compare two null terminated strings, return >0 / 0 / <0 in A, works like
1131:           ; strcmp. The routine expects two pointer in HL and DE which will be
1131:           ; preserved.
1131:           ;
1131: D5        strcmp      push    de
1132: E5                    push    hl
1133: 1A        strcmp_loop ld      a, (de)
1134: FE00                  cp      0                   ; End of first string reached?
1136: 2807                  jr      z, strcmp_exit
1138: BE                    cp      (hl)                ; Compare two characters
1139: 2004                  jr      nz, strcmp_exit     ; Different -> exit
113B: 23                    inc     hl                  ; Prepare comparing the next
113C: 13                    inc     de                  ; characters
113D: 18F4                  jr      strcmp_loop
113F: 96        strcmp_exit sub     (hl)
1140: E1                    pop     hl
1141: D1                    pop     de
1142: C9                    ret
1143:           ;
1143:           ; Convert a single character contained in A to upper case:
1143:           ;
1143: FE61      to_upper    cp      'a'             ; Nothing to do if not lower case
```

```
1145: D8                        ret     c
1146: FE7B                      cp      'z' + 1         ; > 'z'?
1148: D0                        ret     nc              ; Nothing to do, either
1149: E65F                      and     $5f             ; Convert to upper case
114B: C9                        ret
114C:           ;
114C:           ;********************************************************************************
114C:           ;***
114C:           ;*** IO routines
114C:           ;***
114C:           ;********************************************************************************
114C:           ;
114C:           ; Send a CR/LF pair:
114C:           ;
114C: F5        crlf            push    af
114D: 3E0D                      ld      a, cr
114F: CD4012                    call    putc
1152: 3E0A                      ld      a, lf
1154: CD4012                    call    putc
1157: F1                        pop     af
1158: C9                        ret
1159:           ;
1159:           ;  Read a single character from the serial line, result is in A. This is a
1159:           ; blocking system call - it will wait for a character to read! If a non
1159:           ; blocking getc is needed, call getc_nowait instead.
1159:           ;
1159: DB05      getc            in      a, (uart_register_5)
115B: CB47                      bit     0, a
115D: 28FA                      jr      z, getc         ; Wait until there is a character
115F: DB00      getc_nowait     in      a, (uart_register_0)
1161: FE19                      cp      ctrl_y          ; Was it a CTRL-Y?
1163: CA6711                    jp      z, getc_ctrl_y  ; Yes, reset stack pointer and
1166: C9                        ret
1167: 313BFB    getc_ctrl_y     ld      sp, start_type - $1
116A: 217311                    ld      hl, ctrl_y_msg  ; HL is no longer needed since we will
116D:                                                   ; reenter the main loop
116D: CD4B12                    call    puts
1170: C35500                    jp      main_loop       ; reenter the monitor without
1173:                                                   ; printing a welcome message
1173: 0D0A092A
1177: 2A2A2049
117B: 4E544552
117F: 52555054
1183: 202A2A2A
1187: 0D0A00    ctrl_y_msg      defb    cr, lf, tab, "*** INTERRUPT ***", cr, lf, eos
118A:           ;
118A:           ;  Get a byte in hexadecimal notation. The result is returned in A. Since
118A:           ; the routine get_nibble is used only valid characters are accepted - the
118A:           ; input routine only accepts characters 0-9a-f.
118A:           ;
118A: C5        get_byte        push    bc              ; Save contents of B (and C)
118B: CD9D11                    call    get_nibble      ; Get upper nibble
118E: CB07                      rlc     a
1190: CB07                      rlc     a
1192: CB07                      rlc     a
1194: CB07                      rlc     a
1196: 47                        ld      b, a            ; Save upper four bits
1197: CD9D11                    call    get_nibble      ; Get lower nibble
119A: B0                        or      b               ; Combine both nibbles
119B: C1                        pop     bc              ; Restore B (and C)
119C: C9                        ret
119D:           ;
119D:           ;  Get a hexadecimal digit from the serial line. This routine blocks until
119D:           ; a valid character (0-9a-f) has been entered. A valid digit will be echoed
119D:           ; to the serial line interface. The lower 4 bits of A contain the value of
119D:           ; that particular digit.
119D:           ;
119D: CD5911    get_nibble      call    getc            ; Read a character
11A0: CD4311                    call    to_upper        ; Convert to upper case
11A3: CDF610                    call    is_hex          ; Was it a hex digit?
11A6: 30F5                      jr      nc, get_nibble  ; No, get another character
11A8: CD1311                    call    nibble2val      ; Convert nibble to value
11AB: CD2312                    call    print_nibble
11AE: C9                        ret
```

```
11AF:           ;
11AF:           ;  Get a word (16 bit) in hexadecimal notation. The result is returned in HL.
11AF:           ; Since the routines get_byte and therefore get_nibble are called, only valid
11AF:           ; characters (0-9a-f) are accepted.
11AF:           ;
11AF: F5        get_word        push    af
11B0: CD8A11                    call    get_byte        ; Get the upper byte
11B3: 67                        ld      h, a
11B4: CD8A11                    call    get_byte        ; Get the lower byte
11B7: 6F                        ld      l, a
11B8: F1                        pop     af
11B9: C9                        ret
11BA:           ;
11BA:           ;  Read a string from STDIN - HL contains the buffer start address,
11BA:           ; B contains the buffer length. This routine handles the backspace character
11BA:           ; correctly and will echo deleted characters enclosed in backspaces (as it
11BA:           ; was customary with real Teletypes used as terminals). If there are more
11BA:           ; backspaces entered than there are characters to be deleted, a BEL character
11BA:           ; is sent and no further action is taken.
11BA:           ;
11BA: F5        gets            push    af
11BB: C5                        push    bc
11BC: D5                        push    de
11BD: E5                        push    hl
11BE: 0E00                      ld      c, 0                    ; Input mode
11C0: 545D                      ld      de, hl                  ; Remember start of buffer
11C2: CD5911    gets_loop       call    getc                    ; Get a single character
11C5: FE0D                      cp      cr                      ; Skip CR characters
11C7: 28F9                      jr      z, gets_loop            ; only LF will terminate input
11C9: FE08                      cp      bs                      ; Is it a backspace?
11CB: 2021                      jr      nz, gets_1              ; No
11CD: 79                        ld      a, c                    ; In which mode are we?
11CE: FE00                      cp      0
11D0: 2007                      jr      nz, gets_bs             ; Already in backspace
11D2: 0E01                      ld      c, 1                    ; First time in backspace mode
11D4: 3E2F                      ld      a, '/'                  ; Print a slash
11D6: CD4012                    call    putc
11D9: E5        gets_bs         push    hl                      ; Remember HL (needed soon)
11DA: A7                        and     a                       ; Clear carry
11DB: ED52                      sbc     hl, de                  ; Are we at the buffer start?
11DD: E1                        pop     hl                      ; Restore HL
11DE: 2007                      jr      nz, gets_del            ; Not at start, delete char.
11E0: 3E07                      ld      a, bel                  ; Too many backspaces,
11E2: CD4012                    call    putc                    ; ring the bell
11E5: 18DB                      jr      gets_loop               ; and read the next character
11E7: 2B        gets_del        dec     hl                      ; Delete one character
11E8: 7E                        ld      a, (hl)                 ; Get character from buffer
11E9: CD4012                    call    putc                    ; and echo it
11EC: 18D4                      jr      gets_loop               ; Read next character
11EE: F5        gets_1          push    af                      ; Remember character
11EF: 79                        ld      a, c                    ; Did we come from backspace?
11F0: FE00                      cp      0
11F2: 2807                      jr      z, gets_2               ; No
11F4: 0E00                      ld      c, 0                    ; Yes - reset mode
11F6: 3E5C                      ld      a, '\'                  ; Print backslash
11F8: CD4012                    call    putc
11FB: F1        gets_2          pop     af                      ; Restore character
11FC: CD4012                    call    putc                    ; Echo character
11FF: FE0A                      cp      lf                      ; Terminate string at
1201: 2808                      jr      z, gets_exit            ; LF or
1203: FE0D                      cp      cr                      ; CR?
1205: 2804                      jr      z, gets_exit
1207: 77                        ld      (hl), a                 ; Copy character to buffer
1208: 23                        inc     hl
1209: 10B7                      djnz    gets_loop
120B: 3600      gets_exit       ld      (hl), 0                 ; Insert termination byte
120D: E1                        pop     hl
120E: D1                        pop     de
120F: C1                        pop     bc
1210: F1                        pop     af
1211: C9                        ret
1212:           ;
1212:           ;  print_byte prints a single byte in hexadecimal notation to the serial line.
1212:           ; The byte to be printed is expected to be in A.
```

```
1212:           ;
1212: F5        print_byte      push    af              ; Save the contents of the registers
1213: C5                        push    bc
1214: 47                        ld      b, a
1215: 0F                        rrca
1216: 0F                        rrca
1217: 0F                        rrca
1218: 0F                        rrca
1219: CD2312                    call    print_nibble    ; Print high nibble
121C: 78                        ld      a, b
121D: CD2312                    call    print_nibble    ; Print low nibble
1220: C1                        pop     bc              ; Restore original register contents
1221: F1                        pop     af
1222: C9                        ret
1223:           ;
1223:           ;  print_nibble prints a single hex nibble which is contained in the lower
1223:           ; four bits of A:
1223:           ;
1223: F5        print_nibble    push    af              ; We won't destroy the contents of A
1224: E60F                      and     $f              ; Just in case...
1226: C630                      add     '0'             ; If we have a digit we are done here.
1228: FE3A                      cp      '9' + 1         ; Is the result > 9?
122A: 3802                      jr      c, print_nibble_1
122C: C607                      add     'A' - '0' - $a  ; Take care of A-F
122E: CD4012    print_nibble_1  call    putc            ; Print the nibble and
1231: F1                        pop     af              ; restore the original value of A
1232: C9                        ret
1233:           ;
1233:           ;  print_word prints the four hex digits of a word to the serial line. The
1233:           ; word is expected to be in HL.
1233:           ;
1233: E5        print_word      push    hl
1234: F5                        push    af
1235: 7C                        ld      a, h
1236: CD1212                    call    print_byte
1239: 7D                        ld      a, l
123A: CD1212                    call    print_byte
123D: F1                        pop     af
123E: E1                        pop     hl
123F: C9                        ret
1240:           ;
1240:           ; Send a single character to the serial line (a contains the character):
1240:           ;
1240: F5        putc            push    af
1241: DB05      tx_ready_loop   in      a, (uart_register_5)
1243: CB6F                      bit     5, a
1245: 28FA                      jr      z, tx_ready_loop
1247: F1                        pop af
1248: D300                      out     (uart_register_0), a
124A: C9                        ret
124B:           ;
124B:           ; Send a string to the serial line, HL contains the pointer to the string:
124B:           ;
124B: F5        puts            push    af
124C: E5                        push    hl
124D: 7E        puts_loop       ld      a, (hl)
124E: FE00                      cp      eos             ; End of string reached?
1250: 2806                      jr      z, puts_end     ; Yes
1252: CD4012                    call    putc
1255: 23                        inc     hl              ; Increment character pointer
1256: 18F5                      jr      puts_loop       ; Transmit next character
1258: E1        puts_end        pop     hl
1259: F1                        pop     af
125A: C9                        ret
125B:           ;
125B:           ; stroup converts a string to upper case
125B:           ;
125B: F5        stroup          push    af
125C: E5                        push    hl
125D: 7E        stroup_loop     ld      a, (hl)         ; Get a character
125E: FE00                      cp      0               ; End of string reached?
1260: 2807                      jr      z, stroup_exit  ; Yes
1262: CD4311                    call    to_upper        ; No, convert to upper case
1265: 77                        ld      (hl), a         ; Write the character back to memory
```

```
1266: 23                        inc     hl              ; Prepare for next character
1267: 18F4                      jr      stroup_loop
1269: E1        stroup_exit     pop     hl
126A: F1                        pop     af
126B: C9                        ret
126C:           ;
126C:           ;  Test the UART status, RX status -> carry flag, TX status -> Z flag
126C:           ; C == 1: A character is available in the buffer.
126C:           ; Z == 1: A character can be sent.
126C:           ;
126C: DB05      uart_status     in      a, (uart_register_5)
126E: 0F                        rrca                            ; Rotate RX status into carry
126F: CB67                      bit     4, a                    ; Check TX status (after rot!)
1271: C9                        ret
1272:           ;
1272:           ;
1272:           ;*******************************************************************************
1272:           ;***
1272:           ;*** IDE routines
1272:           ;***
1272:           ;*******************************************************************************
1272:           ;
1272:           ; Miscellaneous contants:
1272:           ;
1272:           ide_retries     equ     $ff
1272:           ;
1272:           #if N8VEM = 1
1272:           ;
1272:           ;  Control bytes for setting up the 82C55 for reading/writing from/to an
1272:           ; IDE device:
1272:           ;
1272:           ppide_rd        equ     $92                     ; CTL -> out, LSB/MSB <- in
1272:           ppide_wr        equ     $80                     ; CTL/LSB/MSB -> out
1272:           ;
1272:           ; Constants for IDE control lines:
1272:           ;
1272:           line_ide_a0     equ     $01
1272:           line_ide_a1     equ     $02
1272:           line_ide_a2     equ     $04
1272:           line_ide_cs0    equ     $08
1272:           line_ide_cs1    equ     $10
1272:           line_ide_wr     equ     $20
1272:           line_ide_rd     equ     $40
1272:           line_ide_rst    equ     $80
1272:           ;
1272:           ; Combined constants for various IDE-registers:
1272:           ;
1272:           reg_ide_data    equ     line_ide_cs0
1272:           reg_ide_err     equ     line_ide_cs0                            + line_ide_a0
1272:           ;
1272:           ;       Bit mapping of ide_error_code register:
1272:           ;
1272:           ;               0: 1 = DAM not found
1272:           ;               1: 1 = Track 0 not found
1272:           ;               2: 1 = Command aborted
1272:           ;               3: Reserved
1272:           ;               4: 1 = ID not found
1272:           ;               5: Reserved
1272:           ;               6: 1 = Uncorrectable ECC error
1272:           ;               7: 1 = Bad block detected
1272:           ;
1272:           reg_ide_secnum  equ     line_ide_cs0                + line_ide_a1
1272:           ;
1272:           ;       Typically set to 1 sector to be transferred
1272:           ;
1272:           reg_ide_lba0    equ     line_ide_cs0                + line_ide_a1 + line_ide_a0
1272:           reg_ide_lba1    equ     line_ide_cs0 + line_ide_a2
1272:           reg_ide_lba2    equ     line_ide_cs0 + line_ide_a2                + line_ide_a0
1272:           reg_ide_lba3    equ     line_ide_cs0 + line_ide_a2 + line_ide_a1
1272:           ;
1272:           ;       Bit mapping of ide_lba3 register:
1272:           ;
1272:           ;               0 - 3: LBA bits 24 - 27
1272:           ;               4    : Master (0) or slave (1) selection
```

```
1272:           ;               5    : Always 1
1272:           ;               6    : Set to 1 for LBA access
1272:           ;               7    : Always 1
1272:           ;
1272:           reg_ide_cmd     equ     line_ide_cs0 + line_ide_a2 + line_ide_a1 + line_ide_a0
1272:           reg_ide_status  equ     reg_ide_cmd
1272:           ;
1272:           ;       Useful commands (when written):
1272:           ;
1272:           ;               $20: Read sectors with retry
1272:           ;               $30: Write sectors with retry
1272:           ;               $EC: Identify drive
1272:           ;
1272:           ;       Status bits (when read):
1272:           ;
1272:           ;               0 = ERR:  1 = Previous command resulted in an error
1272:           ;               1 = IDX:  Unused
1272:           ;               2 = CORR: Unused
1272:           ;               3 = DRQ:  1 = Data Request Ready (sector buffer ready)
1272:           ;               4 = DSC:  Unused
1272:           ;               5 = DF:   1 = Write fault
1272:           ;               6 = RDY:  1 = Ready to accept command
1272:           ;               7 = BUSY: 1 = Controller is busy executing a command
1272:           ;
1272:           reg_ide_cntl    equ     line_ide_cs1 + line_ide_a2 + line_ide_a1
1272:           reg_ide_astatus equ     line_ide_cs1 + line_ide_a2 + line_ide_a1 + line_ide_a0
1272:           ;
1272:           ; IDE commands:
1272:           ;
1272:           ide_cmd_recal   equ     $10
1272:           ide_cmd_read    equ     $20
1272:           ide_cmd_write   equ     $30
1272:           ide_cmd_init    equ     $91
1272:           ide_cmd_id      equ     $ec
1272:           ide_cmd_down    equ     $e0
1272:           ide_cmd_spinup  equ     $e1
1272:           ;
1272:           ; IDE routines:
1272:           ;
1272:           ;  Test if the buffer of the IDE disk drive is ready for transfer. If not,
1272:           ; carry will be set, otherwise carry is reset. The contents of register A will
1272:           ; be destroyed!
1272:           ;
1272:           ide_bfr_ready   push    bc
1272:                           and     a                       ; Clear carry assuming no error
1272:                           ld      b, ide_retries          ; How many retries?
1272:           ide_bfr_loop    ld      a, reg_ide_status       ; Prepare reading the IDE SR
1272:                           call    ppide_read              ; Read status register
1272:                           ld      a, c                    ; Get lower 8 bits
1272:                           bit     3, a                    ; Check DRQ bit
1272:                           jr      nz, ide_bfr_exit        ; Buffer is ready
1272:                           push    bc
1272:                           ld      b, $0                   ; Wait a moment
1272:           ide_bfr_wait    nop
1272:                           djnz    ide_bfr_wait
1272:                           pop     bc
1272:                           djnz    ide_bfr_loop            ; Retry
1272:                           scf                             ; Set carry to indicate timeout
1272:                           ld      hl, ide_bfr_rdy_err
1272:                           call    puts
1272:           ide_bfr_exit    pop     bc
1272:                           ret
1272:           ide_bfr_rdy_err defb "FATAL(IDE): ide_bfr_ready timeout!", cr, lf, eos
1272:           ;
1272:           ;  Test if there is any error flagged by the drive. If carry is cleared, no
1272:           ; error occured, otherwise carry will be set. The contents of register A will
1272:           ; be destroyed.
1272:           ;
1272:           ide_error_check and     a                       ; Clear carry (no err expected)
1272:                           ld      a, reg_ide_status       ; Prepare reading the IDE SR
1272:                           call    ppide_read              ; Read the status register
1272:                           ld      a, c                    ; Get lower 8 bits
1272:                           bit     0, a                    ; Test error bit
1272:                           jr      z, ide_ec_exit          ; Everything is OK
```

```
1272:                           scf                             ; Set carry due to error
1272:           ide_ec_exit     ret
1272:           ;
1272:           ;  Get ID information from drive. HL is expected to point to a 512 byte byte
1272:           ; sector buffer. If carry is set, the function did not complete correctly and
1272:           ; was aborted.
1272:           ;
1272:           ide_get_id      push    af
1272:                           push    bc
1272:                           push    hl
1272:                           call    ide_ready               ; Is the drive ready?
1272:                           jr      c, ide_get_id_err       ; No - timeout!
1272:                           ld      c, $a0                  ; Master, no LBA addressing
1272:                           ld      a, reg_ide_lba3         ; Prepare writing LBA3
1272:                           call    ppide_write             ; Perform write access
1272:                           call    ide_ready               ; Did the command complete?
1272:                           jr      c, ide_get_id_err       ; Timeout!
1272:                           ld      c, $ec                  ; Command to read ID
1272:                           ld      a, reg_ide_cmd          ; Prepare writing CMD register
1272:                           call    ppide_write
1272:                           call    ide_ready               ; Can we proceed?
1272:                           jr      c, ide_get_id_err       ; No - timeout, propagate carry
1272:                           call    ide_error_check         ; Any errors?
1272:                           jr      c, ide_get_id_err       ; Yes - something went wrong
1272:                           call    ide_bfr_ready           ; Is the buffer ready to read?
1272:                           jr      c, ide_get_id_err       ; No
1272:                           ld      hl, buffer              ; Load the buffer's address
1272:                           ld      b, $0                   ; We will read 256 words
1272:           ide_get_id_lp   push    bc                      ; PPIDE routines destroy BC!
1272:                           ld      a, reg_ide_data         ; Read 16 bits of data
1272:                           call    ppide_read              ; into BC (MSB/LSB)
1272:                           ld      (hl), b                 ; Store high byte
1272:                           inc     hl                      ; Increment address pointer
1272:                           ld      (hl), c                 ; Store low byte
1272:                           inc     hl                      ; Increment address pointer
1272:                           pop     bc                      ; Restore BC (loop counter)
1272:                           djnz    ide_get_id_lp           ; Read next word
1272:                           jr      ide_get_id_exit         ; Everything OK, just exit
1272:           ide_get_id_err  ld      hl, ide_get_id_msg      ; Print error message
1272:                           call    puts
1272:           ide_get_id_exit pop     hl
1272:                           pop     bc
1272:                           pop     af
1272:                           ret
1272:           ide_get_id_msg  defb    "FATAL(IDE): Aborted!", cr, lf
1272:           ;
1272:           ;  Test if the IDE drive is not busy and ready to accept a command. If it is
1272:           ; ready the carry flag will be reset and the function returns. If a time out
1272:           ; occurs, C will be set prior to returning to the caller. Register A will
1272:           ; be destroyed!
1272:           ;
1272:           ide_ready       push    bc
1272:                           and     a                       ; Clear carry assuming no error
1272:                           ld      b, ide_retries          ; Number of retries to timeout
1272:           ide_ready_loop  ld      a, reg_ide_status       ; Prepare reading the IDE SR
1272:                           call    ppide_read              ; Read status register
1272:                           ld      a, c                    ; Get lower 8 bits
1272:                           and     a, $c0                  ; Only bits 7 and 6 are needed
1272:                           xor     $40                     ; Invert the ready flag
1272:                           jr      z, ide_ready_exit       ; Exit if ready and not busy
1272:                           push    bc
1272:                           ld      b, $0                   ; Wait a moment
1272:           ide_ready_wait  nop
1272:                           djnz    ide_ready_wait
1272:                           pop     bc
1272:                           djnz    ide_ready_loop          ; Retry
1272:                           scf                             ; Set carry due to timeout
1272:                           ld      hl, ide_rdy_error
1272:                           call    puts
1272:                           ld      a, reg_ide_err          ; Prepare reading error code
1272:                           call    ppide_read
1272:                           ld      a, c
1272:                           call    print_byte
1272:           ide_ready_exit  pop     bc
```

```
1272:                           ret
1272:           ide_rdy_error   defb    "FATAL(IDE): ide_ready timeout!", cr, lf, eos
1272:           ;
1272:           ;  Read a sector from the drive. If carry is set after return, the function did
1272:           ; not complete correctly due to a timeout. HL is expected to contain the start
1272:           ; address of the sector buffer while BC and DE contain the sector address
1272:           ; (LBA3, 2, 1 and 0). Register A's contents will be destroyed!
1272:           ;
1272:           ide_rs          push    bc
1272:                           push    hl
1272:                           call    ide_ready               ; Is the drive ready?
1272:                           jr      c, ide_rs_err           ; No - timeout!
1272:                           call    ide_set_lba             ; Setup the drive's registers
1272:                           call    ide_ready               ; Everything OK?
1272:                           jr      c, ide_rs_err           ; No - timeout!
1272:                           ld      c, ide_cmd_read         ; Prepare a read command
1272:                           ld      a, reg_ide_cmd
1272:                           call    ppide_write             ; Issue read command
1272:                           call    ide_ready               ; Can we proceed?
1272:                           jr      c, ide_rs_err           ; No - timeout, set carry
1272:                           call    ide_error_check         ; Any errors?
1272:                           jr      c, ide_rs_err           ; Yes - something went wrong
1272:                           call    ide_bfr_ready           ; Is the buffer ready to read?
1272:                           jr      c, ide_rs_err           ; No
1272:                           ld      b, $0                   ; We will read 256 words
1272:           ide_rs_loop     push    bc                      ; PPIDE-routines destroy BC!
1272:                           ld      a, reg_ide_data         ; Read 16 bits of data
1272:                           call    ppide_read              ; into BC (MSB/LSB)
1272:                           ld      (hl), c                 ; Store low byte
1272:                           inc     hl                      ; Increment address pointer
1272:                           ld      (hl), b                 ; Store high byte
1272:                           inc     hl                      ; Increment pointer
1272:                           pop     bc
1272:                           djnz    ide_rs_loop             ; Read next word until done
1272:                           jr      ide_rs_exit
1272:           ide_rs_err      ld      hl, ide_rs_err_msg      ; Print error message
1272:                           call    puts
1272:           ide_rs_exit     pop     hl
1272:                           pop     bc
1272:                           ret
1272:           ide_rs_err_msg  defb    "FATAL(IDE): ide_rs timeout!", cr, lf, eos
1272:           ;
1272:           ;  Set sector count and LBA registers of the drive. Registers BC and DE contain
1272:           ; the sector address (LBA 3, 2, 1 and 0).
1272:           ;
1272:           ide_set_lba     push    af
1272:                           push    bc                      ; BC will be destroyed
1272:                           push    bc                      ; ...twice!
1272:                           ld      c, $1                   ; We will transfer
1272:                           ld      a, reg_ide_secnum       ; one sector at a time
1272:                           call    ppide_write
1272:                           ld      c, e                    ; Set LBA0
1272:                           ld      a, reg_ide_lba0
1272:                           call    ppide_write
1272:                           ld      c, d                    ; Set LBA1
1272:                           ld      a, reg_ide_lba1
1272:                           call    ppide_write
1272:                           pop     bc                      ; Restore BC to LBA2/3
1272:                           ld      a, reg_ide_lba2         ; Set LBA2
1272:                           call    ppide_write
1272:                           ld      a, b                    ; Special treatment for LBA3
1272:                           and     $0f                     ; Only bits 0 - 3 are LBA3
1272:                           or      $e0                     ; Select LBA and master drive
1272:                           ld      c, a                    ; Set LBA3
1272:                           ld      a, reg_ide_lba3
1272:                           call    ppide_write
1272:                           pop     bc
1272:                           pop     af
1272:                           ret
1272:           ;
1272:           ;  Write a sector to the drive. If carry is set after return, the function did
1272:           ; not complete correctly due to a timeout. HL is expected to contain the start
1272:           ; address of the sector buffer while BC and DE contain the sector address
1272:           ; (LBA3, 2, 1 and 0). Register A's contents will be destroyed!
```

```
1272:           ;
1272:           ide_ws          push    bc
1272:                           push    hl
1272:                           call    ide_ready               ; Is the drive ready?
1272:                           jr      c, ide_ws_err           ; No - timeout!
1272:                           call    ide_set_lba             ; Setup the drive's registers
1272:                           call    ide_ready               ; Everything OK?
1272:                           jr      c, ide_ws_err           ; No - timeout!
1272:                           ld      c, ide_cmd_write        ; Prepare write command
1272:                           ld      a, reg_ide_cmd
1272:                           call    ppide_write             ; Execute read command
1272:                           call    ide_ready               ; Can we proceed?
1272:                           jr      c, ide_ws_err           ; No - timeout, set carry
1272:                           call    ide_error_check         ; Any errors?
1272:                           jr      c, ide_ws_err           ; Yes - something went wrong
1272:                           call    ide_bfr_ready           ; Is the buffer ready to read?
1272:                           jr      c, ide_ws_err           ; No
1272:                           ld      b, $0                   ; We will write 256 words
1272:           ide_ws_loop     push    bc                      ; BC will be destroyed
1272:                           ld      c, (hl)                 ; Read low byte
1272:                           inc     hl                      ; Increment pointer
1272:                           ld      b, (hl)                 ; Read high byte
1272:                           inc     hl
1272:                           ld      a, reg_ide_data         ; Prepare writing to data reg.
1272:                           call    ppide_write
1272:                           pop     bc
1272:                           djnz    ide_ws_loop
1272:                           jr      ide_ws_exit
1272:           ide_ws_err      ld      hl, ide_ws_err_msg      ; Print error message
1272:                           call    puts
1272:           ide_ws_exit     pop     hl
1272:                           pop     bc
1272:                           ret
1272:           ide_ws_err_msg  defb    "FATAL(IDE): ide_ws timeout!", cr, lf, eos
1272:           ;
1272:           ; PPIDE-low level IO routines:
1272:           ;
1272:           ;  ppide_read: Read from an IDE controller register. A contains the IDE
1272:           ; register address while B and C will hold the 16 bit read from that
1272:           ; particular register. Please note that asserting the various control and
1272:           ; address lines has to be done sequentially, first asserting the address
1272:           ; bits then setting read, doing the actual read access, deasserting read
1272:           ; and finally clearing the address bits. A and F will not be changed.
1272:           ;
1272:           ;
1272:           ppide_read      push    af                      ; Save register number
1272:                           push    af                      ; ...twice
1272:                           ld      a, ppide_rd             ; Set PPI ports for read acc.
1272:                           out     (reg_ppi_cntl), a       ; Configure PPI
1272:                           pop     af                      ; Restore register number
1272:                           out     (reg_ppide_cntl), a     ; Setup address
1272:                           or      line_ide_rd             ; Assert read signal
1272:                           out     (reg_ppide_cntl), a     ; Send address + read
1272:                           push    af                      ; We will need the reg. again
1272:                           in      a, (reg_ppide_lsb)      ; Read LSB from IDE
1272:                           ld      c, a                    ; Store it into C
1272:                           in      a, (reg_ppide_msb)      ; Read MSB from IDE
1272:                           ld      b, a                    ; Store it into B
1272:                           pop     af                      ; Prepare deassertion
1272:                           xor     line_ide_rd             ; Clear read signal
1272:                           out     (reg_ppide_cntl), a     ; Deassert read
1272:                           xor     a                       ; Clear address bits, too
1272:                           out     (reg_ppide_cntl), a
1272:                           pop     af                      ; Restore original contents
1272:                           ret
1272:           ;
1272:           ;  ppide_write: Perform a write access to an IDE controller register. Register
1272:           ; A contains the desired register address, B and C contain the MSB/LSB of the
1272:           ; value to be written to the IDE controller. A and F will not be changed.
1272:           ;
1272:           ppide_write     push    af                      ; Save register number
1272:                           push    af                      ; ...twice
1272:                           ld      a, ppide_wr             ; Set PPI ports for write acc.
1272:                           out     (reg_ppi_cntl), a       ; Configure PPI
```

```
1272:                           ld      a, c                    ; Get LSB to be written
1272:                           out     (reg_ppide_lsb), a      ; Set PPI lines
1272:                           ld      a, b                    ; Get MSB
1272:                           out     (reg_ppide_msb), a      ; Set PPI lines
1272:                           pop     af                      ; Restore register number
1272:                           out     (reg_ppide_cntl), a     ; Setup address lines
1272:                           or      line_ide_wr             ; Prepare write line
1272:                           out     (reg_ppide_cntl), a     ; Assert write signal
1272:                           xor     line_ide_wr             ; Reset write bit
1272:                           out     (reg_ppide_cntl), a     ; Deassert write signal
1272:                           xor     a                       ; Reset address lines
1272:                           out     (reg_ppide_cntl), a     ; Deassert all signals
1272:                           pop     af                      ; Restore original contents
1272:                           ret
1272:           ;
1272:           #else                                           ; Homebrew Z80 computer
1272:           ;
1272:           ide_data_low    equ     ide_base + $0
1272:           ide_data_high   equ     ide_base + $8
1272:           ide_error_code  equ     ide_base + $1
1272:           ;
1272:           ;       Bit mapping of ide_error_code register:
1272:           ;
1272:           ;               0: 1 = DAM not found
1272:           ;               1: 1 = Track 0 not found
1272:           ;               2: 1 = Command aborted
1272:           ;               3: Reserved
1272:           ;               4: 1 = ID not found
1272:           ;               5: Reserved
1272:           ;               6: 1 = Uncorrectable ECC error
1272:           ;               7: 1 = Bad block detected
1272:           ;
1272:           ide_secnum      equ     ide_base + $2
1272:           ;
1272:           ;       Typically set to 1 sector to be transf.
1272:           ;
1272:           ide_lba0        equ     ide_base + $3
1272:           ide_lba1        equ     ide_base + $4
1272:           ide_lba2        equ     ide_base + $5
1272:           ide_lba3        equ     ide_base + $6
1272:           ;
1272:           ;       Bit mapping of ide_lba3 register:
1272:           ;
1272:           ;               0 - 3: LBA bits 24 - 27
1272:           ;               4    : Master (0) or slave (1) selection
1272:           ;               5    : Always 1
1272:           ;               6    : Set to 1 for LBA access
1272:           ;               7    : Always 1
1272:           ;
1272:           ide_status_cmd  equ     ide_base + $7
1272:           ;
1272:           ;       Useful commands (when written):
1272:           ;
1272:           ;               $20: Read sectors with retry
1272:           ;               $30: Write sectors with retry
1272:           ;               $EC: Identify drive
1272:           ;
1272:           ;       Status bits (when read):
1272:           ;
1272:           ;               0 = ERR:  1 = Previous command resulted in an error
1272:           ;               1 = IDX:  Unused
1272:           ;               2 = CORR: Unused
1272:           ;               3 = DRQ:  1 = Data Request Ready (sector buffer ready)
1272:           ;               4 = DSC:  Unused
1272:           ;               5 = DF:   1 = Write fault
1272:           ;               6 = RDY:  1 = Ready to accept command
1272:           ;               7 = BUSY: 1 = Controller is busy executing a command
1272:           ;
1272:           ;  Test if the buffer of the IDE disk drive is ready for transfer. If not,
1272:           ; carry will be set, otherwise carry is reset. The contents of register A will
1272:           ; be destroyed!
1272:           ;
1272: C5        ide_bfr_ready   push    bc
1273: A7                        and     a                       ; Clear carry assuming no error
```

```
1274: 06FF                      ld      b, ide_retries          ; How many retries?
1276: DB17      ide_bfr_loop    in      a, (ide_status_cmd)     ; Read IDE status register
1278: CB5F                      bit     3, a                    ; Check DRQ bit
127A: 2010                      jr      nz, ide_bfr_exit        ; Buffer is ready
127C: C5                        push    bc
127D: 0600                      ld      b, $0                   ; Wait a moment
127F: 00        ide_bfr_wait    nop
1280: 10FD                      djnz    ide_bfr_wait
1282: C1                        pop     bc
1283: 10F1                      djnz    ide_bfr_loop            ; Retry
1285: 37                        scf                             ; Set carry to indicate timeout
1286: 218E12                    ld      hl, ide_bfr_rdy_err
1289: CD4B12                    call    puts
128C: C1        ide_bfr_exit    pop     bc
128D: C9                        ret
128E: 46415441
1292: 4C284944
1296: 45293A20
129A: 6964655F
129E: 6266725F
12A2: 72656164
12A6: 79207469
12AA: 6D656F75
12AE: 74210D0A
12B2: 00        ide_bfr_rdy_err defb "FATAL(IDE): ide_bfr_ready timeout!", cr, lf, eos
12B3:           ;
12B3:           ;  Test if there is any error flagged by the drive. If carry is cleared, no
12B3:           ; error occured, otherwise carry will be set. The contents of register A will
12B3:           ; be destroyed.
12B3:           ;
12B3: A7        ide_error_check and     a                       ; Clear carry (no err expected)
12B4: DB17                      in      a, (ide_status_cmd)     ; Read status register
12B6: CB47                      bit     0, a                    ; Test error bit
12B8: 2801                      jr      z, ide_ec_exit          ; Everything is OK
12BA: 37                        scf                             ; Set carry due to error
12BB: C9        ide_ec_exit     ret
12BC:           ;
12BC:           ;  Get ID information from drive. HL is expected to point to a 512 byte byte
12BC:           ; sector buffer. If carry is set, the function did not complete correctly and
12BC:           ; was aborted.
12BC:           ;
12BC: F5        ide_get_id      push    af
12BD: C5                        push    bc
12BE: E5                        push    hl
12BF: CD1213                    call    ide_ready               ; Is the drive ready?
12C2: 382E                      jr      c, ide_get_id_err       ; No - timeout!
12C4: 3EA0                      ld      a, $a0                  ; Master, no LBA addressing
12C6: D316                      out     (ide_lba3), a
12C8: CD1213                    call    ide_ready               ; Did the command complete?
12CB: 3825                      jr      c, ide_get_id_err       ; Timeout!
12CD: 3EEC                      ld      a, $ec                  ; Command to read ID
12CF: D317                      out     (ide_status_cmd), a     ; Write command to drive
12D1: CD1213                    call    ide_ready               ; Can we proceed?
12D4: 381C                      jr      c, ide_get_id_err       ; No - timeout, propagate carry
12D6: CDB312                    call    ide_error_check         ; Any errors?
12D9: 3817                      jr      c, ide_get_id_err       ; Yes - something went wrong
12DB: CD7212                    call    ide_bfr_ready           ; Is the buffer ready to read?
12DE: 3812                      jr      c, ide_get_id_err       ; No
12E0: 2100FE                    ld      hl, buffer              ; Load the buffer's address
12E3: 0600                      ld      b, $0                   ; We will read 256 words
12E5: DB10      ide_get_id_lp   in      a, (ide_data_low)       ; Read high (!) byte
12E7: 4F                        ld      c, a
12E8: DB18                      in      a, (ide_data_high)      ; Read low (!) byte
12EA: 77                        ld      (hl), a
12EB: 23                        inc     hl
12EC: 71                        ld      (hl), c
12ED: 23                        inc     hl
12EE: 10F5                      djnz    ide_get_id_lp           ; Read next word
12F0: 1806                      jr      ide_get_id_exit         ; Everything OK, just exit
12F2: 21FC12    ide_get_id_err  ld      hl, ide_get_id_msg      ; Print error message
12F5: CD4B12                    call    puts
12F8: E1        ide_get_id_exit pop     hl
12F9: C1                        pop     bc
12FA: F1                        pop     af
```

```
12FB: C9                    ret
12FC: 46415441
1300: 4C284944
1304: 45293A20
1308: 41626F72
130C: 74656421
1310: 0D0A      ide_get_id_msg  defb    "FATAL(IDE): Aborted!", cr, lf
1312:           ;
1312:           ;  Test if the IDE drive is not busy and ready to accept a command. If it is
1312:           ; ready the carry flag will be reset and the function returns. If a time out
1312:           ; occurs, C will be set prior to returning to the caller. Register A will
1312:           ; be destroyed!
1312:           ;
1312: C5        ide_ready       push    bc
1313: A7                        and     a                       ; Clear carry assuming no error
1314: 06FF                      ld      b, ide_retries          ; Number of retries to timeout
1316: DB17      ide_ready_loop  in      a, (ide_status_cmd)     ; Read drive status
1318: E6C0                      and     a, $c0                  ; Only bits 7 and 6 are needed
131A: EE40                      xor     $40                     ; Invert the ready flag
131C: 2815                      jr      z, ide_ready_exit       ; Exit if ready and not busy
131E: C5                        push    bc
131F: 0600                      ld      b, $0                   ; Wait a moment
1321: 00        ide_ready_wait  nop
1322: 10FD                      djnz    ide_ready_wait
1324: C1                        pop     bc
1325: 10EF                      djnz    ide_ready_loop          ; Retry
1327: 37                        scf                             ; Set carry due to timeout
1328: 213513                    ld      hl, ide_rdy_error
132B: CD4B12                    call    puts
132E: DB11                      in      a, (ide_error_code)
1330: CD1212                    call    print_byte
1333: C1        ide_ready_exit  pop     bc
1334: C9                        ret
1335: 46415441
1339: 4C284944
133D: 45293A20
1341: 6964655F
1345: 72656164
1349: 79207469
134D: 6D656F75
1351: 74210D0A
1355: 00        ide_rdy_error   defb    "FATAL(IDE): ide_ready timeout!", cr, lf, eos
1356:           ;
1356:           ;  Read a sector from the drive. If carry is set after return, the function did
1356:           ; not complete correctly due to a timeout. HL is expected to contain the start
1356:           ; address of the sector buffer while BC and DE contain the sector address
1356:           ; (LBA3, 2, 1 and 0). Register A's contents will be destroyed!
1356:           ;
1356: C5        ide_rs          push    bc
1357: E5                        push    hl
1358: CD1213                    call    ide_ready               ; Is the drive ready?
135B: 3829                      jr      c, ide_rs_err           ; No - timeout!
135D: CDAD13                    call    ide_set_lba             ; Setup the drive's registers
1360: CD1213                    call    ide_ready               ; Everything OK?
1363: 3821                      jr      c, ide_rs_err           ; No - timeout!
1365: 3E20                      ld      a, $20
1367: D317                      out     (ide_status_cmd), a     ; Issue read command
1369: CD1213                    call    ide_ready               ; Can we proceed?
136C: 3818                      jr      c, ide_rs_err           ; No - timeout, set carry
136E: CDB312                    call    ide_error_check         ; Any errors?
1371: 3813                      jr      c, ide_rs_err           ; Yes - something went wrong
1373: CD7212                    call    ide_bfr_ready           ; Is the buffer ready to read?
1376: 380E                      jr      c, ide_rs_err           ; No
1378: 0600                      ld      b, $0                   ; We will read 256 words
137A: DB10      ide_rs_loop     in      a, (ide_data_low)       ; Read low byte
137C: 77                        ld      (hl), a                 ; Store this byte
137D: 23                        inc     hl
137E: DB18                      in      a, (ide_data_high)      ; Read high byte
1380: 77                        ld      (hl), a
1381: 23                        inc     hl
1382: 10F6                      djnz    ide_rs_loop             ; Read next word until done
1384: 1806                      jr      ide_rs_exit
1386: 218F13    ide_rs_err      ld      hl, ide_rs_err_msg      ; Print error message
1389: CD4B12                    call    puts
```

```
138C: E1        ide_rs_exit     pop     hl
138D: C1                        pop     bc
138E: C9                        ret
138F: 46415441
1393: 4C284944
1397: 45293A20
139B: 6964655F
139F: 72732074
13A3: 696D656F
13A7: 7574210D
13AB: 0A00      ide_rs_err_msg  defb    "FATAL(IDE): ide_rs timeout!", cr, lf, eos
13AD:           ;
13AD:           ;  Set sector count and LBA registers of the drive. Registers BC and DE contain
13AD:           ; the sector address (LBA 3, 2, 1 and 0).
13AD:           ;
13AD: F5        ide_set_lba     push    af
13AE: 3E01                      ld      a, $1                   ; We will transfer
13B0: D312                      out     (ide_secnum), a         ; one sector at a time
13B2: 7B                        ld      a, e
13B3: D313                      out     (ide_lba0), a           ; Set LBA0, 1 and 2 directly
13B5: 7A                        ld      a, d
13B6: D314                      out     (ide_lba1), a
13B8: 79                        ld      a, c
13B9: D315                      out     (ide_lba2), a
13BB: 78                        ld      a, b                    ; Special treatment for LBA3
13BC: E60F                      and     $0f                     ; Only bits 0 - 3 are LBA3
13BE: F6E0                      or      $e0                     ; Select LBA and master drive
13C0: D316                      out     (ide_lba3), a
13C2: F1                        pop     af
13C3: C9                        ret
13C4:           ;
13C4:           ;  Write a sector from the drive. If carry is set after return, the function did
13C4:           ; not complete correctly due to a timeout. HL is expected to contain the start
13C4:           ; address of the sector buffer while BC and DE contain the sector address
13C4:           ; (LBA3, 2, 1 and 0). Register A's contents will be destroyed!
13C4:           ;
13C4: C5        ide_ws          push    bc
13C5: E5                        push    hl
13C6: CD1213                    call    ide_ready               ; Is the drive ready?
13C9: 382A                      jr      c, ide_ws_err           ; No - timeout!
13CB: CDAD13                    call    ide_set_lba             ; Setup the drive's registers
13CE: CD1213                    call    ide_ready               ; Everything OK?
13D1: 3822                      jr      c, ide_ws_err           ; No - timeout!
13D3: 3E30                      ld      a, $30
13D5: D317                      out     (ide_status_cmd), a     ; Issue read command
13D7: CD1213                    call    ide_ready               ; Can we proceed?
13DA: 3819                      jr      c, ide_ws_err           ; No - timeout, set carry
13DC: CDB312                    call    ide_error_check         ; Any errors?
13DF: 3814                      jr      c, ide_ws_err           ; Yes - something went wrong
13E1: CD7212                    call    ide_bfr_ready           ; Is the buffer ready to read?
13E4: 380F                      jr      c, ide_ws_err           ; No
13E6: 0600                      ld      b, $0                   ; We will write 256 word
13E8: 7E        ide_ws_loop     ld      a, (hl)                 ; Get first byte from memory
13E9: 4F                        ld      c, a
13EA: 23                        inc     hl
13EB: 7E                        ld      a, (hl)                 ; Get next byte
13EC: D318                      out     (ide_data_high), a      ; Write high byte to controller
13EE: 79                        ld      a, c                    ; Recall low byte again
13EF: D310                      out     (ide_data_low), a       ; Write low byte -> strobe
13F1: 10F5                      djnz    ide_ws_loop
13F3: 1806                      jr      ide_ws_exit
13F5: 21FE13    ide_ws_err      ld      hl, ide_ws_err_msg      ; Print error message
13F8: CD4B12                    call    puts
13FB: E1        ide_ws_exit     pop     hl
13FC: C1                        pop     bc
13FD: C9                        ret
13FE: 46415441
1402: 4C284944
1406: 45293A20
140A: 6964655F
140E: 77732074
1412: 696D656F
1416: 7574210D
141A: 0A00      ide_ws_err_msg  defb    "FATAL(IDE): ide_ws timeout!", cr, lf, eos
```

```
141C:           ;
141C:           #endif                                          ; N8VEM = 1?
141C:           ;
141C:           ;==============================================================================
141C:           ;***
141C:           ;*** Miscellaneous functions
141C:           ;***
141C:           ;==============================================================================
141C:           ;
141C:           ; Clear the computer (not to be called - jump into this routine):
141C:           ;
141C: 213CFB    cold_start      ld      hl, start_type
141F: 3600                      ld      (hl), $00
1421: 212D14    warm_start      ld      hl, clear_msg
1424: CD4B12                    call    puts
1427: 3E00                      ld      a, $00
1429: 32FFFF                    ld      (ram_end), a
142C: C7                        rst     $00
142D: 434C4541
1431: 520D0A00  clear_msg       defb    "CLEAR", cr, lf, eos
1435:           ;
1435:           ;
1435:           ;==============================================================================
1435:           ;***
1435:           ;*** Mathematical routines
1435:           ;***
1435:           ;==============================================================================
1435:           ;
1435:           ; 32 bit add routine from
1435:           ;       http://www.andreadrian.de/oldcpu/Z80_number_cruncher.html
1435:           ;
1435:           ; ADD ROUTINE 32+32BIT=32BIT
1435:           ; H'L'HL = H'L'HL + D'E'DE
1435:           ; CHANGES FLAGS
1435:           ;
1435: 19        ADD32:  ADD     HL,DE   ; 16-BIT ADD OF HL AND DE
1436: D9                EXX
1437: ED5A              ADC     HL,DE   ; 16-BIT ADD OF HL AND DE WITH CARRY
1439: D9                EXX
143A: C9                RET
143B:           ;
143B:           ; 32 bit multiplication routine from
143B:           ;       http://www.andreadrian.de/oldcpu/Z80_number_cruncher.html
143B:           ;
143B:           ; MULTIPLY ROUTINE 32*32BIT=32BIT
143B:           ; H'L'HL = B'C'BC * D'E'DE; NEEDS REGISTER A, CHANGES FLAGS
143B:           ;
143B: A7        MUL32:  AND     A               ; RESET CARRY FLAG
143C: ED62              SBC     HL,HL           ; LOWER RESULT = 0
143E: D9                EXX
143F: ED62              SBC     HL,HL           ; HIGHER RESULT = 0
1441: 78                LD      A,B             ; MPR IS AC'BC
1442: 0620              LD      B,32            ; INITIALIZE LOOP COUNTER
1444:           MUL32LOOP:
1444: CB2F              SRA     A               ; RIGHT SHIFT MPR
1446: CB19              RR      C
1448: D9                EXX
1449: CB18              RR      B
144B: CB19              RR      C               ; LOWEST BIT INTO CARRY
144D: 3005              JR      NC,MUL32NOADD
144F: 19                ADD     HL,DE           ; RESULT += MPD
1450: D9                EXX
1451: ED5A              ADC     HL,DE
1453: D9                EXX
1454:           MUL32NOADD:
1454: CB23              SLA     E               ; LEFT SHIFT MPD
1456: CB12              RL      D
1458: D9                EXX
1459: CB13              RL      E
145B: CB12              RL      D
145D: 10E5              DJNZ    MUL32LOOP
145F: D9                EXX
1460: C9                RET
1461:           ;
```

```
1461:           ;*******************************************************************************
1461:           ;***
1461:           ;*** FAT file system routines
1461:           ;***
1461:           ;*******************************************************************************
1461:           ;
1461:           ;  Read a single byte from a file. IY points to the FCB. The byte read is
1461:           ; returned in A, on EOF the carry flag will be set.
1461:           ;
1461: C5        fgetc           push    bc
1462: D5                        push    de
1463: E5                        push    hl
1464:           ;  Check if fcb_file_pointer == fcb_file_size. In this case we have reached
1464:           ; EOF and will return with a set carry bit. (As a side effect, the attempt to
1464:           ; read from a file which has not been successfully opened before will be
1464:           ; handled like encountering an EOF at the first fgetc call.)
1464: FD7E0C                    ld      a, (iy + fcb_file_size)
1467: FDBE13                    cp      (iy + fcb_file_pointer)
146A: 201C                      jr      nz, fgetc_start
146C: FD7E0D                    ld      a, (iy + fcb_file_size + 1)
146F: FDBE14                    cp      (iy + fcb_file_pointer + 1)
1472: 2014                      jr      nz, fgetc_start
1474: FD7E0E                    ld      a, (iy + fcb_file_size + 2)
1477: FDBE15                    cp      (iy + fcb_file_pointer + 2)
147A: 200C                      jr      nz, fgetc_start
147C: FD7E0F                    ld      a, (iy + fcb_file_size + 3)
147F: FDBE16                    cp      (iy + fcb_file_pointer + 3)
1482: 2004                      jr      nz, fgetc_start
1484:           ; We have reached EOF, so set carry and leave this routine:
1484: 37                        scf
1485: C37115                    jp      fgetc_exit
1488:           ;  Check if the lower 9 bits of the file pointer are zero. In this case
1488:           ; we need to read another sector (maybe from another cluster):
1488: FD7E13    fgetc_start     ld      a, (iy + fcb_file_pointer)
148B: FE00                      cp      0
148D: C23D15                    jp      nz, fgetc_getc          ; Bits 0-7 are not zero
1490: FD7E14                    ld      a, (iy + fcb_file_pointer + 1)
1493: E601                      and     1
1495: C23D15                    jp      nz, fgetc_getc          ; Bit 8 is not zero
1498:           ; The file_pointer modulo 512 is zero, so we have to load the next sector:
1498:           ; We have to check if fcb_current_cluster == 0 which will be the case in the
1498:           ; initial run. Then we will copy fcb_first_cluster into fcb_current_cluster.
1498: FD7E17                    ld      a, (iy + fcb_current_cluster)
149B: FE00                      cp      0
149D: 2015                      jr      nz, fgetc_continue      ; Not the initial case
149F: FD7E18                    ld      a, (iy + fcb_current_cluster + 1)
14A2: FE00                      cp      0
14A4: 200E                      jr      nz, fgetc_continue      ; Not the initial case
14A6:           ; Initial case: We have to fill fcb_current_cluster with fcb_first_cluste:
14A6: FD7E10                    ld      a, (iy + fcb_first_cluster)
14A9: FD7717                    ld      (iy + fcb_current_cluster), a
14AC: FD7E11                    ld      a, (iy + fcb_first_cluster + 1)
14AF: FD7718                    ld      (iy + fcb_current_cluster + 1), a
14B2: 1837                      jr      fgetc_clu2sec
14B4:           ; Here is the normal case - we will check if fcb_cluster_sector is zero -
14B4:           ; in this case we have to determine the next sector to be loaded by looking
14B4:           ; up the FAT. Otherwise (fcb_cluster_sector != 0) we will just get the next
14B4:           ; sector in the current cluster.
14B4: FD7E1D    fgetc_continue  ld      a, (iy + fcb_cluster_sector)
14B7: 2043                      jr      nz, fgetc_same          ; The current cluster is valid
14B9:           ;  Here we know that we need the first sector of the next cluster of the file.
14B9:           ; The upper eight bits of the fcb_current_cluster point to the sector of the
14B9:           ; FAT where the entry we are looking for is located (this is true since a
14B9:           ; sector contains 512 bytes which corresponds to 256 FAT entries). So we must
14B9:           ; load the sector with the number fatstart + fcb_current_cluster[15-8] into
14B9:           ; the IDE buffer and locate the entry with the address
14B9:           ; fcb_current_cluster[7-0] * 2. This entry contains the sector number we are
14B9:           ; looking for.
14B9: 2AF4FD                    ld      hl, (fat1start)
14BC: FD4E18                    ld      c, (iy + fcb_current_cluster + 1)
14BF: 0600                      ld      b, 0
14C1: 09                        add     hl, bc
14C2: 545D                      ld      de, hl                  ; Needed for ide_rs
14C4: 010000                    ld      bc, 0
```

```
14C7: 2AF6FD                    ld      hl, (fat1start + 2)
14CA: ED4A                      adc     hl, bc
14CC: 444D                      ld      bc, hl                  ; Needed for ide_rs
14CE: 2100FE                    ld      hl, buffer
14D1: CD5613                    call    ide_rs
14D4:           ;  Now the sector containing the FAT entry we are looking for is available in
14D4:           ; the IDE buffer. Now we need fcb_current_cluster[7-0] * 2
14D4: 0600                      ld      b, 0
14D6: FD4E17                    ld      c, (iy + fcb_current_cluster)
14D9: CB21                      sla     c
14DB: CB10                      rl      b
14DD:           ; Now get the entry:
14DD: 2100FE                    ld      hl, buffer
14E0: 09                        add     hl, bc
14E1: 4E23462B                  ld      bc, (hl)
14E5: FD7117                    ld      (iy + fcb_current_cluster), c
14E8: FD7017                    ld      (iy + fcb_current_cluster), b
14EB:           ; Now we determine the first sector of the cluster to be read:
14EB: 3AE5FD    fgetc_clu2sec   ld      a, (clusiz)             ; Initialize fcb_cluster_sector
14EE: FD771D                    ld      (iy + fcb_cluster_sector), a
14F1: FD6E17                    ld      l, (iy + fcb_current_cluster)
14F4: FD6618                    ld      h, (iy + fcb_current_cluster + 1)
14F7: CD3818                    call    clu2sec                 ; Convert cluster to sector
14FA: 1826                      jr      fgetc_rs
14FC: A7        fgetc_same      and     a                       ; Clear carry
14FD: 010100                    ld      bc, 1                   ; Increment fcb_current_sector
1500: FD6E19                    ld      l, (iy + fcb_current_sector)
1503: FD661A                    ld      h, (iy + fcb_current_sector + 1)
1506: 09                        add     hl, bc
1507: FD7519                    ld      (iy + fcb_current_sector), l
150A: 5D                        ld      e, l                    ; Needed for ide_rs
150B: FD741A                    ld      (iy + fcb_current_sector + 1), h
150E: 54                        ld      d, h                    ; Needed for ide_rs
150F: FD6E1B                    ld      l, (iy + fcb_current_sector + 2)
1512: FD661C                    ld      h, (iy + fcb_current_sector + 3)
1515: 010000                    ld      bc, 0
1518: ED4A                      adc     hl, bc
151A: FD751B                    ld      (iy + fcb_current_sector + 2), l
151D: 4D                        ld      c, l                    ; Needed for ide_rs
151E: FD741C                    ld      (iy + fcb_current_sector + 3), h
1521: 44                        ld      b, h                    ; Neede for ide_rs
1522: FD7319    fgetc_rs        ld      (iy + fcb_current_sector), e    ; Now read the sector
1525: FD721A                    ld      (iy + fcb_current_sector + 1), d
1528: FD711B                    ld      (iy + fcb_current_sector + 2), c
152B: FD701C                    ld      (iy + fcb_current_sector + 3), b
152E:           ; Let HL point to the sector buffer in the FCB:
152E: FDE5                      push    iy                      ; Start of FCB
1530: E1                        pop     hl
1531: C5                        push    bc
1532: 011E00                    ld      bc, fcb_file_buffer     ; Displacement of sector buffer
1535: 09                        add     hl, bc
1536: C1                        pop     bc
1537: CD5613                    call    ide_rs                  ; Read a single sector from disk
153A:           ; Since we have read a sector we have to decrement fcb_cluster_sector
153A: FD351D                    dec     (iy + fcb_cluster_sector)
153D:           ; Here we read and return a single character from the sector buffer:
153D: FDE5      fgetc_getc      push    iy
153F: E1                        pop     hl                      ; Copy IY to HL
1540: 011E00                    ld      bc, fcb_file_buffer
1543: 09                        add     hl, bc                  ; HL points to the sector bfr.
1544:           ; Get the lower 9 bits of the file pointer as displacement for the buffer:
1544: FD4E13                    ld      c, (iy + fcb_file_pointer)
1547: FD7E14                    ld      a, (iy + fcb_file_pointer + 1)
154A: E601                      and     1                       ; Get rid of bits 9-15
154C: 47                        ld      b, a
154D: 09                        add     hl, bc                  ; Add byte offset
154E: 7E                        ld      a, (hl)                 ; get one byte from buffer
154F:           ; Increment the file pointer:
154F: FD6E13                    ld      l, (iy + fcb_file_pointer)
1552: FD6614                    ld      h, (iy + fcb_file_pointer + 1)
1555: 010100                    ld      bc, 1
1558: 09                        add     hl, bc
1559: FD7513                    ld      (iy + fcb_file_pointer), l
155C: FD7414                    ld      (iy + fcb_file_pointer + 1), h
```

```
155F: 010000                  ld      bc, 0
1562: FD6E15                  ld      l, (iy + fcb_file_pointer + 2)
1565: FD6616                  ld      h, (iy + fcb_file_pointer + 3)
1568: ED4A                    adc     hl, bc
156A: FD7515                  ld      (iy + fcb_file_pointer + 2), l
156D: FD7416                  ld      (iy + fcb_file_pointer + 3), h
1570:           ;
1570: A7                      and     a                       ; Clear carry
1571: E1        fgetc_exit    pop     hl
1572: D1                      pop     de
1573: C1                      pop     bc
1574: C9                      ret
1575:           ;
1575:           ;  Clear the FCB to which IY points -- this should be called every time one
1575:           ; creates a new FCB. (Please note that fopen does its own call to clear_fcb.)
1575:           ;
1575: F5        clear_fcb     push    af                      ; We have to save so many
1576: C5                      push    bc                      ; Registers since the FCB is
1577: D5                      push    de                      ; cleared using LDIR.
1578: E5                      push    hl
1579: 3E00                    ld      a, 0
157B: FDE5                    push    iy
157D: E1                      pop     hl
157E: 77                      ld      (hl), a                 ; Clear first byte of FCB
157F: 545D                    ld      de, hl
1581: 13                      inc     de
1582: 011E00                  ld      bc, fcb_file_buffer
1585: EDB0                    ldir                            ; And transfer this zero byte
1587: E1                      pop     hl                      ; down to the relevant rest
1588: D1                      pop     de                      ; of the buffer.
1589: C1                      pop     bc
158A: F1                      pop     af
158B: C9                      ret
158C:           ;
158C:           ; Dump a file control block (FCB) - the start address is expected in IY.
158C:           ;
158C: F5        dump_fcb      push    af
158D: E5                      push    hl
158E: 211E16                  ld      hl, dump_fcb_1
1591: CD4B12                  call    puts
1594: FDE5                    push    iy                      ; Load HL with
1596: E1                      pop     hl                      ; the contents of IY
1597: CD3312                  call    print_word
159A:           ; Print the filename:
159A: 213716                  ld      hl, dump_fcb_2
159D: CD4B12                  call    puts
15A0: FDE5                    push    iy
15A2: E1                      pop     hl
15A3: CD4B12                  call    puts
15A6:           ; Print file size:
15A6: 214C16                  ld      hl, dump_fcb_3
15A9: CD4B12                  call    puts
15AC: FD660F                  ld      h, (iy + fcb_file_size + 3)
15AF: FD6E0E                  ld      l, (iy + fcb_file_size + 2)
15B2: CD3312                  call    print_word
15B5: FD660D                  ld      h, (iy + fcb_file_size + 1)
15B8: FD6E0C                  ld      l, (iy + fcb_file_size)
15BB: CD3312                  call    print_word
15BE:           ; Print cluster number:
15BE: 216116                  ld      hl, dump_fcb_4
15C1: CD4B12                  call    puts
15C4: FD6611                  ld      h, (iy + fcb_first_cluster + 1)
15C7: FD6E10                  ld      l, (iy + fcb_first_cluster)
15CA: CD3312                  call    print_word
15CD:           ; Print file type:
15CD: 217616                  ld      hl, dump_fcb_5
15D0: CD4B12                  call    puts
15D3: FD7E12                  ld      a, (iy + fcb_file_type)
15D6: CD1212                  call    print_byte
15D9:           ; Print file pointer:
15D9: 218B16                  ld      hl, dump_fcb_6
15DC: CD4B12                  call    puts
15DF: FD6616                  ld      h, (iy + fcb_file_pointer + 3)
15E2: FD6E15                  ld      l, (iy + fcb_file_pointer + 2)
```

```
15E5: CD3312                  call    print_word
15E8: FD6614                  ld      h, (iy + fcb_file_pointer + 1)
15EB: FD6E13                  ld      l, (iy + fcb_file_pointer)
15EE: CD3312                  call    print_word
15F1:           ; Print current cluster number:
15F1: 21A016                  ld      hl, dump_fcb_7
15F4: CD4B12                  call    puts
15F7: FD6618                  ld      h, (iy + fcb_current_cluster + 1)
15FA: FD6E17                  ld      l, (iy + fcb_current_cluster)
15FD: CD3312                  call    print_word
1600:           ; Print current sector:
1600: 21B516                  ld      hl, dump_fcb_8
1603: CD4B12                  call    puts
1606: FD661C                  ld      h, (iy + fcb_current_sector + 3)
1609: FD6E1B                  ld      l, (iy + fcb_current_sector + 2)
160C: CD3312                  call    print_word
160F: FD661A                  ld      h, (iy + fcb_current_sector + 1)
1612: FD6E19                  ld      l, (iy + fcb_current_sector)
1615: CD3312                  call    print_word
1618: CD4C11                  call    crlf
161B: E1                      pop     hl
161C: F1                      pop     af
161D: C9                      ret
161E: 44756D70
1622: 206F6620
1626: 46434220
162A: 61742061
162E: 64647265
1632: 73733A20
1636: 00        dump_fcb_1    defb    "Dump of FCB at address: ", eos
1637: 0D0A0946
163B: 696C6520
163F: 6E616D65
1643: 20202020
1647: 20203A20
164B: 00        dump_fcb_2    defb    cr, lf, tab, "File name      : ", eos
164C: 0D0A0946
1650: 696C6520
1654: 73697A65
1658: 20202020
165C: 20203A20
1660: 00        dump_fcb_3    defb    cr, lf, tab, "File size      : ", eos
1661: 0D0A0931
1665: 73742063
1669: 6C757374
166D: 65722020
1671: 20203A20
1675: 00        dump_fcb_4    defb    cr, lf, tab, "1st cluster    : ", eos
1676: 0D0A0946
167A: 696C6520
167E: 74797065
1682: 20202020
1686: 20203A20
168A: 00        dump_fcb_5    defb    cr, lf, tab, "File type      : ", eos
168B: 0D0A0946
168F: 696C6520
1693: 706F696E
1697: 74657220
169B: 20203A20
169F: 00        dump_fcb_6    defb    cr, lf, tab, "File pointer   : ", eos
16A0: 0D0A0943
16A4: 75727265
16A8: 6E742063
16AC: 6C757374
16B0: 65723A20
16B4: 00        dump_fcb_7    defb    cr, lf, tab, "Current cluster: ", eos
16B5: 0D0A0943
16B9: 75727265
16BD: 6E742073
16C1: 6563746F
16C5: 72203A20
16C9: 00        dump_fcb_8    defb    cr, lf, tab, "Current sector : ", eos
16CA:           ;
16CA:           ;  Convert a user specified filename to an 8.3-filename without dot and
```

```
16CA:           ; with terminating null byte. HL points to the input string, DE points to
16CA:           ; a 12 character buffer for the filename. This function is used by
16CA:           ; fopen which expects a human readable string that will be transformed into
16CA:           ; an 8.3-filename without the dot for the following directory lookup.
16CA:           ;
16CA: F5        str2filename    push    af
16CB: C5                        push    bc
16CC: D5                        push    de
16CD: E5                        push    hl
16CE: ED535DFB                  ld      (str2filename_de), de
16D2: 3E20                      ld      a, ' '                  ; Initialize output buffer
16D4: 060B                      ld      b, $b                   ; Fill 11 bytes with spaces
16D6: 12        str2filiniloop  ld      (de), a
16D7: 13                        inc     de
16D8: 10FC                      djnz    str2filiniloop
16DA: 3E00                      ld      a, 0                    ; Add terminating null byte
16DC: 12                        ld      (de), a
16DD: ED5B5DFB                  ld      de, (str2filename_de)   ; Restore DE pointer
16E1:           ; Start string conversion
16E1: 0608                      ld      b, 8
16E3: 7E        str2filini_nam  ld      a, (hl)
16E4: FE00                      cp      0                       ; End of string reached?
16E6: 282F                      jr      z, str2filini_x
16E8: FE2E                      cp      '.'                     ; Dot found?
16EA: 2812                      jr      z, str2filini_ext
16EC: 12                        ld      (de), a
16ED: 13                        inc     de
16EE: 23                        inc     hl
16EF: 05                        dec     b
16F0: 20F1                      jr      nz, str2filini_nam
16F2: 7E        str2filini_skip ld      a, (hl)
16F3: FE00                      cp      0                       ; End of string without dot?
16F5: 2820                      jr      z, str2filini_x         ; Nothing more to do
16F7: FE2E                      cp      '.'
16F9: 2803                      jr      z, str2filini_ext       ; Take care of extension
16FB: 23                        inc     hl                      ; Prepare for next character
16FC: 18F4                      jr      str2filini_skip         ; Skip more characters
16FE: 23        str2filini_ext  inc     hl                      ; Skip the dot
16FF: E5                        push    hl                      ; Make sure DE points
1700: 2A5DFB                    ld      hl, (str2filename_de)   ; into the filename buffer
1703: 010800                    ld      bc, 8                   ; at the start position
1706: 09                        add     hl, bc                  ; of the filename extension
1707: 545D                      ld      de, hl
1709: E1                        pop     hl
170A: 0603                      ld      b, 3
170C: 7E        str2filini_elp  ld      a, (hl)
170D: FE00                      cp      0                       ; End of string reached?
170F: 2806                      jr      z, str2filini_x         ; Nothing more to do
1711: 12                        ld      (de), a
1712: 13                        inc     de
1713: 23                        inc     hl
1714: 05                        dec     b
1715: 20F5                      jr      nz, str2filini_elp      ; Next extension character
1717: E1        str2filini_x    pop     hl
1718: D1                        pop     de
1719: C1                        pop     bc
171A: F1                        pop     af
171B: C9                        ret
171C:           ;
171C:           ;  Open a file with given filename (format: 'FFFFFFFFXXX') in the root directory
171C:           ; and return the 1st cluster number for that file. If the file can not
171C:           ; be found, $0000 will be returned in the FCB.
171C:           ;  At entry, HL must point to the string buffer while IY points to a valid
171C:           ; file control block that will hold all necessary data for future file accesses.
171C:           ; In addition to that DE must point to a 12 character string buffer.
171C:           ;
171C: F5        fopen           push    af
171D: C5                        push    bc
171E: D5                        push    de
171F: E5                        push    hl
1720: DDE5                      push    ix
1722: 2255FB                    ld      (fopen_scr), hl
1725: 21DCFD                    ld      hl, fatname             ; Check if a disk has been
1728: 7E                        ld      a, (hl)                 ; mounted.
```

```
1729: FE00                  cp      0
172B: CAD417                jp      z, fopen_e1             ; No disk - error exit
172E: CD7515                call    clear_fcb
1731: FDE5                  push    iy                      ; Copy IY to DE
1733: D1                    pop     de
1734: 2A55FB                ld      hl, (fopen_scr)         ; Create the filename
1737: CDCA16                call    str2filename            ; Convert string to a filename
173A: 2100FE                ld      hl, buffer              ; Compute buffer overflow
173D: 010002                ld      bc, $0200               ; address - this is the bfr siz.
1740: 09                    add     hl, bc                  ; and will be used in the loop
1741: 225BFB                ld      (fopen_eob), hl         ; This is the buffer end addr.
1744:           ;
1744: 2AF8FD                ld      hl, (rootstart)         ; Remember the initial root
1747: 2257FB                ld      (fopen_rsc), hl         ; sector number
174A: 2AFAFD                ld      hl, (rootstart + 2)
174D: 2259FB                ld      (fopen_rsc + 2), hl
1750:           ; Read one root directory sector
1750: ED4B59FB  fopen_nbf   ld      bc, (fopen_rsc + 2)
1754: ED5B57FB              ld      de, (fopen_rsc)
1758: 2100FE                ld      hl, buffer
175B: CD5613                call    ide_rs                  ; Read one sector
175E: DAD917                jp      c, fopen_e2             ; Exit on read error
1761: 2255FB    fopen_lp    ld      (fopen_scr), hl
1764: AF                    xor     a                       ; Last entry?
1765: BE                    cp      (hl)                    ; The last entry has first
1766: CADF17                jp      z, fopen_x              ; byte = $0
1769: 3EE5                  ld      a, $e5                  ; Deleted entry?
176B: BE                    cp      (hl)
176C: 284B                  jr      z, fopen_nxt            ; Get next entry
176E:           ;           ld      (fopen_scr), hl
176E: DD2A55FB              ld      ix, (fopen_scr)
1772: DD7E0B                ld      a, (ix + $b)            ; Get attribute byte
1775: FE0F                  cp      $0f
1777: 2840                  jr      z, fopen_nxt            ; Skip long name
1779: CB67                  bit     4, a                    ; Skip directories
177B: 203C                  jr      nz, fopen_nxt
177D:           ; Compare the filename with the one we are looking for:
177D: DD360B00              ld      (ix + $b), 0            ; Clear attribute byte
1781: ED5B55FB              ld      de, (fopen_scr)
1785: FDE5                  push    iy                      ; Prepare string comparison
1787: E1                    pop     hl
1788: CD3111                call    strcmp                  ; Compare filename with string
178B: FE00                  cp      0                       ; Are strings equal?
178D: 202A                  jr      nz, fopen_nxt           ; No - check next entry
178F: DD7E1B                ld      a, (ix + $1a + 1)       ; Read cluster number and
1792:           ; Save cluster_number into fcb_first_cluster:
1792: FD7711                ld      (iy + fcb_first_cluster + 1), a
1795: DD7E1A                ld      a, (ix + $1a)
1798: FD7710                ld      (iy + fcb_first_cluster), a
179B: DD7E1C                ld      a, (ix + $1c)           ; Save file size to FCB
179E: FD770C                ld      (iy + fcb_file_size), a
17A1: DD7E1D                ld      a, (ix + $1d)           ; Save file size to FCB
17A4: FD770D                ld      (iy + fcb_file_size + 1), a
17A7: DD7E1E                ld      a, (ix + $1e)           ; Save file size to FCB
17AA: FD770E                ld      (iy + fcb_file_size + 2), a
17AD: DD7E1F                ld      a, (ix + $1f)           ; Save file size to FCB
17B0: FD770F                ld      (iy + fcb_file_size + 3), a
17B3: FD361201              ld      (iy + fcb_file_type), 1 ; Set file type to found
17B7: 1826                  jr      fopen_x                 ; Terminate lookup loop
17B9: 012000    fopen_nxt   ld      bc, $20
17BC: 2A55FB                ld      hl, (fopen_scr)
17BF: 09                    add     hl, bc
17C0: 2255FB                ld      (fopen_scr), hl
17C3: ED4B5BFB              ld      bc, (fopen_eob)         ; Check for end of buffer
17C7: A7                    and     a                       ; Clear carry
17C8: ED42                  sbc     hl, bc                  ; ...no 16 bit cp :-(
17CA: C26117                jp      nz, fopen_lp            ; Buffer is still valid
17CD: 2157FB                ld      hl, fopen_rsc           ; Increment sector number
17D0: 34                    inc     (hl)                    ; 16 bits are enough :-)
17D1: C35017                jp      fopen_nbf               ; Read next directory sector
17D4: 21E617    fopen_e1    ld      hl, fopen_nmn           ; No disk mounted
17D7: 1803                  jr      fopen_err               ; Print error message
17D9: 210718    fopen_e2    ld      hl, fopen_rer           ; Directoy sector read error
17DC: CD4B12    fopen_err   call    puts
```

```
17DF: DDE1      fopen_x         pop     ix
17E1: E1                        pop     hl
17E2: D1                        pop     de
17E3: C1                        pop     bc
17E4: F1                        pop     af
17E5: C9                        ret
17E6: 46415441
17EA: 4C28464F
17EE: 50454E29
17F2: 3A204E6F
17F6: 20646973
17FA: 6B206D6F
17FE: 756E7465
1802: 64210D0A
1806: 00        fopen_nmn       defb    "FATAL(FOPEN): No disk mounted!", cr, lf, eos
1807: 46415441
180B: 4C28464F
180F: 50454E29
1813: 3A20436F
1817: 756C6420
181B: 6E6F7420
181F: 72656164
1823: 20646972
1827: 6563746F
182B: 72792073
182F: 6563746F
1833: 7221      fopen_rer       defb    "FATAL(FOPEN): Could not read directory sector!"
1835: 0D0A00                    defb    cr, lf, eos
1838:           ;
1838:           ;  Convert a cluster number into a sector number. The cluster number is
1838:           ; expected in HL, the corresponding sector number will be returned in
1838:           ; BC and DE, thus ide_rs or ide_ws can be called afterwards.
1838:           ;
1838:           ; SECNUM = (CLUNUM - 2) * CLUSIZ + DATASTART
1838:           ;
1838: F5        clu2sec         push    af                      ; Since the 32 bit
1839: E5                        push    hl                      ; multiplication routine
183A: D9                        exx                             ; needs shadow registers
183B: C5                        push    bc                      ; we have to push many,
183C: D5                        push    de                      ; many registers here
183D: E5                        push    hl
183E: 010000                    ld      bc, 0                   ; Clear BC' and DE' for
1841: 5059                      ld      de, bc                  ; 32 bit multiplication
1843: D9                        exx
1844: 010200                    ld      bc, 2                   ; Subtract 2
1847: ED42                      sbc     hl, bc                  ; HL = CLUNUM - 2
1849: 444D                      ld      bc, hl                  ; BC = HL; BC' = 0
184B: 3AE5FD                    ld      a, (clusiz)
184E: 1600                      ld      d, 0                    ; CLUSIZ bits 8 to 15
1850: 5F                        ld      e, a                    ; DE = CLUSIZ
1851: CD3B14                    call    MUL32                   ; HL = (CLUNUM - 2) * CLUSIZ
1854: ED5BFCFD                  ld      de, (datastart)
1858: D9                        exx
1859: ED5BFEFD                  ld      de, (datastart + 2)
185D: D9                        exx
185E: CD3514                    call    ADD32                   ; HL = HL + DATASTART
1861: D9                        exx
1862: E5                        push    hl
1863: D9                        exx
1864: C1                        pop     bc
1865: 545D                      ld      de, hl
1867: D9                        exx
1868: E1                        pop     hl
1869: D1                        pop     de
186A: C1                        pop     bc
186B: D9                        exx
186C: E1                        pop     hl
186D: F1                        pop     af
186E: C9                        ret
186F:           ;
186F:           ; Print a directory listing
186F:           ;
186F: F5        dirlist         push    af
1870: C5                        push    bc
```

```
1871: D5                    push    de
1872: E5                    push    hl
1873: DDE5                  push    ix
1875: 21DCFD                ld      hl, fatname
1878: 7E                    ld      a, (hl)
1879: FE00                  cp      0
187B: CA4219                jp      z, dirlist_nodisk
187E: DD216DFB              ld      ix, string_81_bfr
1882: DD36082E              ld      (ix + 8), '.'           ; Dot between name and extens.
1886: DD360C00              ld      (ix + 12), 0            ; String terminator
188A: 217219                ld      hl, dirlist_0           ; Print title line
188D: CD4B12                call    puts
1890: 2100FE                ld      hl, buffer              ; Compute buffer overflow
1893: 010002                ld      bc, $0200               ; address - this is the bfr siz.
1896: 09                    add     hl, bc
1897: 2251FB                ld      (dirlist_eob), hl       ; This is the buffer end addr.
189A:           ;
189A: 2AF8FD                ld      hl, (rootstart)         ; Remember the initial root
189D: 224DFB                ld      (dirlist_rootsec), hl   ; sector number
18A0: 2AFAFD                ld      hl, (rootstart + 2)
18A3: 224FFB                ld      (dirlist_rootsec + 2), hl
18A6:           ; Read one root directory sector
18A6: ED4B4FFB  dirlist_nbfr    ld      bc, (dirlist_rootsec + 2)
18AA: ED5B4DFB              ld      de, (dirlist_rootsec)
18AE: 2100FE                ld      hl, buffer
18B1: CD5613                call    ide_rs
18B4: DA3D19                jp      c, dirlist_e1
18B7: AF        dirlist_loop    xor     a                       ; Last entry?
18B8: BE                    cp      (hl)                    ; The last entry has first
18B9: CA4819                jp      z, dirlist_exit         ; byte = $0
18BC: 3EE5                  ld      a, $e5                  ; Deleted entry?
18BE: BE                    cp      (hl)
18BF: 2867                  jr      z, dirlist_next
18C1: 2253FB                ld      (dirlist_scratch), hl
18C4: DD2A53FB              ld      ix, (dirlist_scratch)
18C8: DD7E0B                ld      a, (ix + $b)            ; Get attribute byte
18CB: FE0F                  cp      $0f
18CD: 2859                  jr      z, dirlist_next         ; Skip long name
18CF: 116DFB                ld      de, string_81_bfr       ; Prepare for output
18D2: 010800                ld      bc, 8                   ; Copy first eight characters
18D5: EDB0                  ldir
18D7: 13                    inc     de
18D8: 010300                ld      bc, 3                   ; Copy extension
18DB: EDB0                  ldir
18DD:           ;               ld      hl, de
18DD:           ;               ld      (hl), 0                 ; String terminator
18DD: 216DFB                ld      hl, string_81_bfr
18E0: CD4B12                call    puts
18E3: 21471A                ld      hl, dirlist_NODIR       ; Flag directories with "DIR"
18E6: CB67                  bit     4, a
18E8: 2803                  jr      z, dirlist_prtdir
18EA: 21411A                ld      hl, dirlist_DIR
18ED: CD4B12    dirlist_prtdir  call    puts
18F0: DD661F                ld      h, (ix + $1c + 3)       ; Get and print file size
18F3: DD6E1E                ld      l, (ix + $1c + 2)
18F6: CD3312                call    print_word
18F9: DD661D                ld      h, (ix + $1c + 1)
18FC: DD6E1C                ld      l, (ix + $1c)
18FF: CD3312                call    print_word
1902:           ; Get and print start sector
1902: 3E09                  ld      a, tab
1904: CD4012                call    putc
1907: DD661B                ld      h, (ix + $1a + 1)       ; Get cluster number
190A: DD6E1A                ld      l, (ix + $1a)
190D: 010000                ld      bc, 0                   ; Is file empty?
1910: A7                    and     a                       ; Clear carry
1911: ED42                  sbc     hl, bc                  ; Empty file -> Z set
1913: 280D                  jr      z, dirlist_nosize
1915: CD3818                call    clu2sec
1918: 6069                  ld      hl, bc
191A: CD3312                call    print_word
191D: 626B                  ld      hl, de
191F: CD3312                call    print_word
1922: CD4C11    dirlist_nosize  call    crlf
```

```
1925: 2A53FB                    ld      hl, (dirlist_scratch)
1928: 012000    dirlist_next    ld      bc, $20
192B: 09                        add     hl, bc
192C: ED4B51FB                  ld      bc, (dirlist_eob)       ; Check for end of buffer
1930: A7                        and     a
1931: ED42                      sbc     hl, bc
1933: C2B718                    jp      nz, dirlist_loop        ; Buffer is still valid
1936: 214DFB                    ld      hl, dirlist_rootsec
1939: 34                        inc     (hl)
193A: C3A618                    jp      dirlist_nbfr
193D: 210F1A    dirlist_e1      ld      hl, dirlist_1
1940: 1803                      jr      dirlist_x
1942: 214F19    dirlist_nodisk  ld      hl, dirlist_nomnt
1945: CD4B12    dirlist_x       call    puts
1948: DDE1      dirlist_exit    pop     ix
194A: E1                        pop     hl
194B: D1                        pop     de
194C: C1                        pop     bc
194D: F1                        pop     af
194E: C9                        ret
194F: 46415441
1953: 4C284449
1957: 524C4953
195B: 54293A20
195F: 4E6F2064
1963: 69736B20
1967: 6D6F756E
196B: 74656421
196F: 0D0A00    dirlist_nomnt   defb    "FATAL(DIRLIST): No disk mounted!", cr, lf, eos
1972: 44697265
1976: 63746F72
197A: 7920636F
197E: 6E74656E
1982: 74733A0D
1986: 0A        dirlist_0       defb    "Directory contents:", cr, lf
1987: 2D2D2D2D
      ...
19B2: 0D0A                      defb    "----------------------------------------", cr, lf
19B4: 46494C45
19B8: 4E414D45
19BC: 2E455854
19C0: 20204449
19C4: 523F2020
19C8: 2053495A
19CC: 45202842
19D0: 59544553
19D4: 29                        defb    "FILENAME.EXT  DIR?   SIZE (BYTES)"
19D5: 20203153
19D9: 54205345
19DD: 43540D0A                  defb    "  1ST SECT", cr, lf
19E1: 2D2D2D2D
      ...
1A0C: 0D0A                      defb    "----------------------------------------", cr, lf
1A0E: 00                        defb    eos
1A0F: 46415441
1A13: 4C284449
1A17: 524C4953
1A1B: 54293A20
1A1F: 436F756C
1A23: 64206E6F
1A27: 74207265
1A2B: 61642064
1A2F: 69726563
1A33: 746F7279
1A37: 20736563
1A3B: 746F72    dirlist_1       defb    "FATAL(DIRLIST): Could not read directory sector"
1A3E: 0D0A00                    defb    cr, lf, eos
1A41: 09444952
1A45: 0900      dirlist_DIR     defb    tab, "DIR", tab, eos
1A47: 090900    dirlist_NODIR   defb    tab, tab, eos
1A4A:           ;
1A4A:           ; Perform a disk mount
1A4A:           ;
1A4A: F5        fatmount        push    af
```

```
1A4B: C5                    push    bc
1A4C: D5                    push    de
1A4D: E5                    push    hl
1A4E: DDE5                  push    ix
1A50: 2100FE                ld      hl, buffer              ; Read MBR into buffer
1A53: 010000                ld      bc, 0
1A56: 110000                ld      de, 0
1A59: CD5613                call    ide_rs
1A5C: DAC21B                jp      c, fatmount_e1          ; Error reading MBR?
1A5F: DD21FEFF              ld      ix, buffer + $1fe       ; Check for $55AA as MBR trailer
1A63: 3E55                  ld      a, $55
1A65: DDBE00                cp      (ix)
1A68: C2C71B                jp      nz, fatmount_e2
1A6B: 3EAA                  ld      a, $aa
1A6D: DDBE01                cp      (ix + 1)
1A70: C2C71B                jp      nz, fatmount_e2
1A73: 010800                ld      bc, 8                   ; Get partition start and size
1A76: 21C6FF                ld      hl, buffer + $1c6
1A79: 11ECFD                ld      de, pstart
1A7C: EDB0                  ldir
1A7E: 2100FE                ld      hl, buffer              ; Read partition boot block
1A81: ED5BECFD              ld      de, (pstart)
1A85: ED4BEEFD              ld      bc, (pstart + 2)
1A89: CD5613                call    ide_rs
1A8C: DACC1B                jp      c, fatmount_e3          ; Error reading boot block?
1A8F: 010800                ld      bc, 8                   ; Copy FAT name
1A92: 2103FE                ld      hl, buffer + 3
1A95: 11DCFD                ld      de, fatname
1A98: EDB0                  ldir
1A9A: DD2100FE              ld      ix, buffer
1A9E: 3E02                  ld      a, 2                    ; Check for two FATs
1AA0: DDBE10                cp      (ix + $10)
1AA3: C2D11B                jp      nz, fatmount_e4         ; Wrong number of FATs
1AA6: AF                    xor     a                       ; Check for 512 bytes / sector
1AA7: DDBE0B                cp      (ix + $b)
1AAA: C2D61B                jp      nz, fatmount_e5
1AAD: 3E02                  ld      a, 2
1AAF: DDBE0C                cp      (ix + $c)
1AB2: C2D61B                jp      nz, fatmount_e5
1AB5: 3A0DFE                ld      a, (buffer + $d)        ; Get cluster size
1AB8: 32E5FD                ld      (clusiz), a
1ABB: ED4B0EFE              ld      bc, (buffer + $e)       ; Get reserved sector number
1ABF: ED43E6FD              ld      (ressec), bc
1AC3: ED4B16FE              ld      bc, (buffer + $16)      ; Get FAT size in sectors
1AC7: ED43E8FD              ld      (fatsec), bc
1ACB: ED4B11FE              ld      bc, (buffer + $11)      ; Get length of root directory
1ACF: ED43EAFD              ld      (rootlen), bc
1AD3: 2AECFD                ld      hl, (pstart)            ; Compute
1AD6: ED4BE6FD              ld      bc, (ressec)            ; FAT1START = PSTART + RESSEC
1ADA: 09                    add     hl, bc
1ADB: 22F4FD                ld      (fat1start), hl
1ADE: 2AEEFD                ld      hl, (pstart + 2)
1AE1: 010000                ld      bc, 0
1AE4: ED4A                  adc     hl, bc
1AE6: 22F6FD                ld      (fat1start + 2), hl
1AE9: 2AE8FD                ld      hl, (fatsec)            ; Compute ROOTSTART for two FATs
1AEC: 29                    add     hl, hl                  ; ROOTSTART = FAT1START +
1AED: 444D                  ld      bc, hl                  ;             2 * FATSIZ
1AEF: 2AF4FD                ld      hl, (fat1start)
1AF2: 09                    add     hl, bc
1AF3: 22F8FD                ld      (rootstart), hl
1AF6: 2AF6FD                ld      hl, (fat1start + 2)
1AF9: 010000                ld      bc, 0
1AFC: ED4A                  adc     hl, bc
1AFE: 22FAFD                ld      (rootstart + 2), hl
1B01: ED4BEAFD              ld      bc, (rootlen)           ; Compute rootlen / 16
1B05: CB28                  sra     b                       ; By shifting it four places
1B07: CB19                  rr      c                       ; to the right
1B09: CB28                  sra     b                       ; This value will be used
1B0B: CB19                  rr      c                       ; for the calculation of
1B0D: CB28                  sra     b                       ; DATASTART
1B0F: CB19                  rr      c
1B11: CB28                  sra     b
1B13: CB19                  rr      c
```

```
1B15: 2AF8FD                ld      hl, (rootstart)         ; Computer DATASTART
1B18: 09                    add     hl, bc
1B19: 22FCFD                ld      (datastart), hl
1B1C: 2AFAFD                ld      hl, (rootstart + 2)
1B1F: 010000                ld      bc, 0
1B22: ED4A                  adc     hl, bc
1B24: 22FEFD                ld      (datastart + 2), hl
1B27: 21C21C                ld      hl, fatmount_s1         ; Print mount summary
1B2A: CD4B12                call    puts
1B2D: 21DCFD                ld      hl, fatname
1B30: CD4B12                call    puts
1B33: 21CD1C                ld      hl, fatmount_s2
1B36: CD4B12                call    puts
1B39: 3AE5FD                ld      a, (clusiz)
1B3C: CD1212                call    print_byte
1B3F: 21D91C                ld      hl, fatmount_s3
1B42: CD4B12                call    puts
1B45: 2AE6FD                ld      hl, (ressec)
1B48: CD3312                call    print_word
1B4B: 21E51C                ld      hl, fatmount_s4
1B4E: CD4B12                call    puts
1B51: 2AE8FD                ld      hl, (fatsec)
1B54: CD3312                call    print_word
1B57: 21F11C                ld      hl, fatmount_s5
1B5A: CD4B12                call    puts
1B5D: 2AEAFD                ld      hl, (rootlen)
1B60: CD3312                call    print_word
1B63: 21FE1C                ld      hl, fatmount_s6
1B66: CD4B12                call    puts
1B69: 2AF2FD                ld      hl, (psiz + 2)
1B6C: CD3312                call    print_word
1B6F: 2AF0FD                ld      hl, (psiz)
1B72: CD3312                call    print_word
1B75: 21091D                ld      hl, fatmount_s7
1B78: CD4B12                call    puts
1B7B: 2AEEFD                ld      hl, (pstart + 2)
1B7E: CD3312                call    print_word
1B81: 2AECFD                ld      hl, (pstart)
1B84: CD3312                call    print_word
1B87: 21151D                ld      hl, fatmount_s8
1B8A: CD4B12                call    puts
1B8D: 2AF6FD                ld      hl, (fat1start + 2)
1B90: CD3312                call    print_word
1B93: 2AF4FD                ld      hl, (fat1start)
1B96: CD3312                call    print_word
1B99: 21241D                ld      hl, fatmount_s9
1B9C: CD4B12                call    puts
1B9F: 2AFAFD                ld      hl, (rootstart + 2)
1BA2: CD3312                call    print_word
1BA5: 2AF8FD                ld      hl, (rootstart)
1BA8: CD3312                call    print_word
1BAB: 21331D                ld      hl, fatmount_sa
1BAE: CD4B12                call    puts
1BB1: 2AFEFD                ld      hl, (datastart + 2)
1BB4: CD3312                call    print_word
1BB7: 2AFCFD                ld      hl, (datastart)
1BBA: CD3312                call    print_word
1BBD: CD4C11                call    crlf
1BC0: 181A                  jr      fatmount_exit
1BC2: 21E31B    fatmount_e1 ld      hl, fatmount_1
1BC5: 1812                  jr      fatmount_x
1BC7: 210A1C    fatmount_e2 ld      hl, fatmount_2
1BCA: 180D                  jr      fatmount_x
1BCC: 212A1C    fatmount_e3 ld      hl, fatmount_3
1BCF: 1808                  jr      fatmount_x
1BD1: 21611C    fatmount_e4 ld      hl, fatmount_4
1BD4: 1803                  jr      fatmount_x
1BD6: 218E1C    fatmount_e5 ld      hl, fatmount_5
1BD9: CD4B12    fatmount_x  call    puts
1BDC: DDE1      fatmount_exit pop   ix
1BDE: E1                    pop     hl
1BDF: D1                    pop     de
1BE0: C1                    pop     bc
1BE1: F1                    pop     af
```

```
1BE2: C9                      ret
1BE3: 46415441
1BE7: 4C284641
1BEB: 544D4F55
1BEF: 4E54293A
1BF3: 20436F75
1BF7: 6C64206E
1BFB: 6F742072
1BFF: 65616420
1C03: 4D425221
1C07: 0D0A00    fatmount_1    defb    "FATAL(FATMOUNT): Could not read MBR!", cr, lf, eos
1C0A: 46415441
1C0E: 4C284641
1C12: 544D4F55
1C16: 4E54293A
1C1A: 20496C6C
1C1E: 6567616C
1C22: 204D4252
1C26: 210D0A00  fatmount_2    defb    "FATAL(FATMOUNT): Illegal MBR!", cr, lf, eos
1C2A: 46415441
1C2E: 4C284641
1C32: 544D4F55
1C36: 4E54293A
1C3A: 20436F75
1C3E: 6C64206E
1C42: 6F742072
1C46: 65616420
1C4A: 70617274
1C4E: 6974696F
1C52: 6E20626F
1C56: 6F742062
1C5A: 6C6F636B  fatmount_3    defb    "FATAL(FATMOUNT): Could not read partition boot block"
1C5E: 0D0A00                  defb    cr, lf, eos
1C61: 46415441
1C65: 4C284641
1C69: 544D4F55
1C6D: 4E54293A
1C71: 20464154
1C75: 206E756D
1C79: 62657220
1C7D: 6E6F7420
1C81: 65717561
1C85: 6C207477
1C89: 6F21      fatmount_4    defb    "FATAL(FATMOUNT): FAT number not equal two!"
1C8B: 0D0A00                  defb    cr, lf, eos
1C8E: 46415441
1C92: 4C284641
1C96: 544D4F55
1C9A: 4E54293A
1C9E: 20536563
1CA2: 746F7220
1CA6: 73697A65
1CAA: 206E6F74
1CAE: 20657175
1CB2: 616C2035
1CB6: 31322062
1CBA: 79746573
1CBE: 21        fatmount_5    defb    "FATAL(FATMOUNT): Sector size not equal 512 bytes!"
1CBF: 0D0A00                  defb    cr, lf, eos
1CC2: 09464154
1CC6: 4E414D45
1CCA: 3A0900    fatmount_s1   defb    tab, "FATNAME:", tab, eos
1CCD: 0D0A0943
1CD1: 4C555349
1CD5: 5A3A0900  fatmount_s2   defb    cr, lf, tab, "CLUSIZ:", tab, eos
1CD9: 0D0A0952
1CDD: 45535345
1CE1: 433A0900  fatmount_s3   defb    cr, lf, tab, "RESSEC:", tab, eos
1CE5: 0D0A0946
1CE9: 41545345
1CED: 433A0900  fatmount_s4   defb    cr, lf, tab, "FATSEC:", tab, eos
1CF1: 0D0A0952
1CF5: 4F4F544C
1CF9: 454E3A09
```

```
1CFD: 00        fatmount_s5     defb    cr, lf, tab, "ROOTLEN:", tab, eos
1CFE: 0D0A0950
1D02: 53495A3A
1D06: 090900    fatmount_s6     defb    cr, lf, tab, "PSIZ:", tab, tab, eos
1D09: 0D0A0950
1D0D: 53544152
1D11: 543A0900  fatmount_s7     defb    cr, lf, tab, "PSTART:", tab, eos
1D15: 0D0A0946
1D19: 41543153
1D1D: 54415254
1D21: 3A0900    fatmount_s8     defb    cr, lf, tab, "FAT1START:", tab, eos
1D24: 0D0A0952
1D28: 4F4F5453
1D2C: 54415254
1D30: 3A0900    fatmount_s9     defb    cr, lf, tab, "ROOTSTART:", tab, eos
1D33: 0D0A0944
1D37: 41544153
1D3B: 54415254
1D3F: 3A0900    fatmount_sa     defb    cr, lf, tab, "DATASTART:", tab, eos
1D42:           ;
1D42:           ;  Dismount a FAT volume (invalidate the FAT control block by setting the
1D42:           ; first byte (of fatname) to zero.
1D42:           ;
1D42: F5        fatunmount      push    af
1D43: E5                        push    hl
1D44: AF                        xor     a
1D45: 21DCFD                    ld      hl, fatname
1D48: 77                        ld      (hl), a                 ; Clear first byte of fatname
1D49: E1                        pop     hl
1D4A: F1                        pop     af
1D4B: C9                        ret
1D4C:           ;
1D4C:           ;  Here the dispatch table for calling system routines starts. Every entry
1D4C:           ; must contain only the destination address (2 bytes).
1D4C:           ;
1D4C: 1C14      dispatch_table  defw    cold_start      ; $00 = clear etc.
1D4E:                           ; Parameters:    N/A
1D4E:                           ; Action:        Performs a cold start (memory is cleared!)
1D4E:                           ; Return values: N/A
1D4E:                           ;
1D4E: F610                      defw    is_hex
1D50:                           ; Parameters:    A contains a character code
1D50:                           ; Action:        Tests ('0' <= A <= '9) || ('A' <= A <= 'F')
1D50:                           ; Return values: Carry bit is set if A contains a hex char.
1D50:                           ;
1D50: 0A11                      defw    is_print
1D52:                           ; Parameters:    A contains a charater code
1D52:                           ; Action:        Tests if the character is printable
1D52:                           ; Return values: Carry bit is set if A contains a valid char.
1D52:                           ;
1D52: 4311                      defw    to_upper
1D54:                           ; Parameters:    A contains a character code
1D54:                           ; Action:        Converts an ASCII character into upper case
1D54:                           ; Return values: Converted character code in A
1D54:                           ;
1D54: 4C11                      defw    crlf
1D56:                           ; Parameters:    N/A
1D56:                           ; Action:        Sends a CR/LF to the serial line
1D56:                           ; Return values: N/A
1D56:                           ;
1D56: 5911                      defw    getc
1D58:                           ; Parameters:    N/A
1D58:                           ; Action:        Reads a character code from the serial line
1D58:                           ; Return values: A contains a character code
1D58:                           ;
1D58: 4012                      defw    putc
1D5A:                           ; Parameters:    A contains a character code
1D5A:                           ; Action:        Sends the character code to the serial line
1D5A:                           ; Return values: N/A
1D5A:                           ;
1D5A: 4B12                      defw    puts
1D5C:                           ; Parameters:    HL contains the address of a 0-terminated
1D5C:                           ;                string
1D5C:                           ; Action:        Send the string to the serial line (excluding
```

```
1D5C:                   ;               the termination byte, of course)
1D5C:                   ; Return values: N/A
1D5C:                   ;
1D5C: 3111              defw    strcmp
1D5E:                   ; Parameters:   HL and DE contain the addresses of two strings
1D5E:                   ; Action:       Compare both strings.
1D5E:                   ; Return values: A contains return value, <0 / 0 / >0
1D5E:                   ;
1D5E: BA11              defw    gets
1D60:                   ; Parameters:   HL contains a buffer address, B contains the
1D60:                   ;               buffer length (including the terminating
1D60:                   ;               null byte!)
1D60:                   ; Action:       Reads a string from STDIN. Terminates when
1D60:                   ;               either the buffer is full or the string is
1D60:                   ;               terminated by CR/LF
1D60:                   ; Return values: N/A
1D60:                   ;
1D60: 6114              defw    fgetc
1D62:                   ; Parameters:   IY (pointer to a valid FCB)
1D62:                   ; Action:       Reads a character from a FAT file
1D62:                   ; Return values: Character in A, if EOF has been encountered,
1D62:                   ;               the carry flag will be set
1D62:                   ;
1D62: 8C15              defw    dump_fcb
1D64:                   ; Parameters:   IY (pointer to a valid FCB)
1D64:                   ; Action:       Prints the contents of the FCB in human
1D64:                   ;               readable format to STDOUT
1D64:                   ; Return values: N/A
1D64:                   ;
1D64: 1C17              defw    fopen
1D66:                   ; Parameters:   HL (points to a buffer containing the file
1D66:                   ;               file name), IY (points to an empty FCB),
1D66:                   ;               DE (points to a 12 character string buffer)
1D66:                   ; Action:       Opens a file for reading
1D66:                   ; Return values: N/A (All information is contained in the FCB)
1D66:                   ;
1D66: 6F18              defw    dirlist
1D68:                   ; Parameters:   N/A (relies on a valid FAT control block)
1D68:                   ; Action:       Writes a directory listing to STDOUT
1D68:                   ; Return values: N/A
1D68:                   ;
1D68: 4A1A              defw    fatmount
1D6A:                   ; Parameters:   N/A (needs the global FAT control block)
1D6A:                   ; Action:       Mounts a disk (populates the FAT CB)
1D6A:                   ; Return values: N/A
1D6A:                   ;
1D6A: 421D              defw    fatunmount
1D6C:                   ; Parameters:   N/A (needs the global FAT control block)
1D6C:                   ; Action:       Invalidates the global FAT control block
1D6C:                   ; Return values: N/A
1D6C: 1E11              defw    strchr
1D6E:                   ; Parameters:   HL contains buffer address, A contains
1D6E:                   ;               character to be searched in buffer
1D6E:                   ; Action:       Look for the first occurrence of a character
1D6E:                   ;               in a string
1D6E:                   ; Return values: Carry bit set if character was found. HL
1D6E:                   ;               points to the address of this character in
1D6E:                   ;               the buffer
1D6E: 6C12              defw    uart_status
1D70:                   ; Parameters:   N/A
1D70:                   ; Action:       Checks the UART status
1D70:                   ; Return values: RX status in carry flag, TX status in Z flag.
1D70:                   ;               Register A and F are modified!
1D70: 5F11              defw    getc_nowait
1D72:                   ; Parameters:   N/A
1D72:                   ; Action:       Reads a character code from the serial line
1D72:                   ;               but does not wait for a character to be there.
1D72:                   ;               This function is usable eg. for a Forth
1D72:                   ;               interpreter etc.
1D72:                   ; Return values: A contains a character code
1D72: 3312              defw    print_word
1D74:                   ; Parameters:   HL contains the 16 bit value to be printed
1D74:                   ; Action:       Prints a 16 bit value in hexadecimal repre-
1D74:                   ;               sentation
```

```
1D74:                           ; Return values: N/A
1D74: 1212                      defw    print_byte
1D76:                           ; Parameters:    A contains the byte to be printed
1D76:                           ; Action:        Prints a byte in hexadecimal notation
1D76:                           ; Return values: N/A
1D76: 5B12                      defw    stroup
1D78:                           ; Parameters:    HL points to the string to be converted
1D78:                           ; Action:        Converts a string to upper case
1D78:                           ; Return values: N/A
1D78: AF11                      defw    get_word
1D7A:                           ; Parameters:    N/A
1D7A:                           ; Action:        Reads a four nibble hex-word from stdin
1D7A:                           ; Return values: HL contains the value read
1D7A:           ;
1D7A:           ;*******************************************************************************
1D7A:           ;***
1D7A:           ;*** From here on various subsystems can be included (like Forth, BASIC etc.).
1D7A:           ;***
1D7A:           ;*******************************************************************************
1D7A:           ;
1D7A: 215334    forth_subsystem ld      hl, forth_msg
1D7D: CD4B12                    call    puts
345B: 00        end_of_monitor  defb    0
;... ...The included CAMEL Forth interpreter resides here... ...
345C:           #end                    ; ((inserted by zasm))
```

ulmann@vaxman.de

webmaster@vaxman.de

28-JUN-2013