dastaZ80 Mark III
User's Manual

Disclaimer

The products described in this manual are intended for educational purposes, and should not be used for controlling any machinery, critical component in life support devices or any system in which failure could result in personal injury if any of the described here products fail.

These products are subject to continuous development and improvement. All information of a technical nature and particulars of the products and its use are given by the author in good faith. However, it is acknowledged that there may be errors or omissions in this manual. Therefore, the author cannot accept any liability for any loss or damage arising from the use of any information or particulars in this manual.

Licenses

Document Conventions

The following conventions are used in this manual:

MUST	MUST denotes that the definition is and absolute requirement.
SHOULD	SHOULD denotes that it is recommended, but that there may exist valid reasons to ignore it.
DEVICE	Device names are displayed in bold all upper case letters, and refer to hardware devices.
command	Operating System command keywords are displayed in bold all lower case letters.
<*text*>	Angle brackets enclose variable information that you MUST supply. In place of <*text*>, substitute the value desired. Do not enter the angle brackets when entering the value.
[*text*]	Square brackets enclose variable information that you COULD supply. They are optional. In place of [text], substitute the value desired. Do not enter the square brackets when entering the value.
`Courier`	Text appearing in the `Courier` font represents information that you type in via the keyboard.
0x14B0	Numbers prefixed by 0x indicate an Hexadecimal value. Unless specified, memory addresses are always expressed in Hexadecimal.
Return	Refers to the key Return in the keyboard.

The SD card is referred as **DISK**.

The Floppy Disk Drive is referred as **DISK** or as **FDD**.

The 80 column text VGA output is referred as **CONSOLE** or as **High Resolution Display**.

The 40 column graphics Composite Video output is referred as **Low Resolution Display**.

The Operating System may be referred as DZOS, dzOS or simply OS.

MEMORY refers to both **ROM** and **RAM**.

Although the word **ROM** is used in this manual, the actual chip used in the dastaZ80 computer is of the type **EEPROM**.

Memory used by the **Low Resolution Display** is referred as **VRAM** (Video RAM).

Related Documentation

- dastaZ80 User's Manual[1]
- dastaZ80 Technical Reference Manual[2]
- dzOS Github Repository[3]
- Software for dzOS Github Repository[4]
- Nascom 2 Microcomputer BASIC Programming Manuals[5]
- Z80 Family CPU User Manual[6]

Contents

1 Introduction

The dastaZ80 is a homebrew computer designed and built following the style of the 8-bit computers of the 80s that I used on those days: Amstrad CPC, Commodore 64 and MSX. The name comes from "d"avid "asta" (my name) and "Z80" (the CPU used).

The idea behind the making of this computer came from an initial wish of writing an operating system (OS) for an 8-bit machine. Not comfortable with writing an OS for an already existing computer like an Amstrad CPC, C64, MSX, etc., due to the complexity of its hardware (or rather my lack of knowledge), I decided to built my own 8-bit computer from scratch, so that I could fully understand the hardware and also influence the design.

The OS written by me for this computer is called *DZOS*, from dastaZ80 OS. Sometimes I spell it as *dzOS*. Haven't made my mind yet.

This manual describes the usage of DZOS running on the dastaZ80 computer.

2 dastaZ80 Overview

The dastaZ80 computer comes in two different formats:

- dastaZ80 (Original), in Acorn Archimedes A3010 case.
- dastaZ80DB (*DB* stands for Desktop Box), in a desktop computer box.

Both are functionally the same, except the dastaZ80DB does not have a keyboard nor a Floppy Disk Drive, and is meant to be operated from an external video terminal (like DEC VT100) or a computer running a terminal program (like Minicom or PuTTY). More details about this configuration in the section Setting up the system.

The dastaZ80 Original was the first version I made, and it is a stand-alone computer. Only things needed to use it are: the computer itself, a 5V/4A power supply, a monitor with VGA connector and optionally a monitor with NTSC Composite video input and (also optionally) a pair of amplified speakers or a monitor with RCA stereo connectors.

The dastaZ80DB requires the same, plus an external serial keyboard. Also, the VGA can be exchanged by a serial monitor.

2.1 dastaZ80 Original

- ZiLOG Z80 microprocessor (CPU) running at 7.3728 MHz
- 64 KB RAM: 16 KB reserved for DZOS, 48 KB available for the user and programms.
- Storage devices: Micro SD Card[1], 3.5" DD/HD Floppy Disk Drive.
- Dual Video output: VGA 80 columns by 25 lines and 16 colours, Composite NTSC 15 colours.
- Stereo Sound: 3 channels (1 tone per channel), 7 octaves, 1 noise channel, envelope generator[2].
- Real-Time Clock: Date and Time backed up with button cell battery.
- Keyboard: Acorn Archimedes A3010 keyboard. 102 keys with 12 function keys, cursor keys and numeric pad.
- Case: Repurposed Acorn Archimedes A3010 case with keyboard.
- Expansion ports: GPIO and ROM Cartridges.

2.2 dastaZ80DB

Same as above, with the only differences:

- No keyboard.
- No Floppy Disk Drive.
- Case: desktop box.

[1] Formatted with FAT32 and containing Disk Image Files formatted with DZFS (dastaZ80 File System.)

[2] Same chip (AY-3-8912 PSG) as used in numerous Arcade machines and in Amstrad CPC, Atari ST, MSX, ZX Spectrum and other home computers.

3 Setting up the system

3.1 dastaZ80 Original

You will only need:

- The dastaZ80 computer.
- A 5 Volts (4 Amp) power supply with a female 2.1mm barrel-style DC connector (positive polarity).
- A Micro SD card.
- A monitor with VGA input.
- A LIR2032 Li-Ion Rechargeable 3.6V button cell battery.
- Optionally:
  - A monitor with NTSC Composite input.
  - A 3.5mm jack to 3 RCA Audio/Video cable[3].

3.2 dastaZ80DB

You will need:

- The dastaZ80DB computer.
- Another computer, with serial port (either USB or RS-232).
  - For USB, you will need a TTL-to-USB cable.
  - For RS-232, you will need a TTL-to-RS-232 converter and a RS-232 null modem cable.
- A 5 Volts (4 Amp) power supply with a female 2.1mm barrel-style DC connector (positive polarity).
- A Micro SD card.
- A 3V CR2016 button cell battery.
- Optionally:
  - A monitor with VGA input.
  - A monitor with NTSC Composite input.
  - A 3.5mm jack to 3 RCA Audio/Video cable[4].

3.3 Lets put it together

1. Insert the battery (LIR2032[5] on the dastaZ80 Original and CR2016 on the dastaZ80DB) in the battery holder. Positive goes up.
2. On a modern PC, format the SD card with FAT32.
3. Create a file of 33 MB on the SD card.
   - For example using Linux terminal: `fallocate -l $((33*1024*1024)) dastaZ80.img`
4. Create a file named *_disks.cfg* in the root of the SD card, and add two lines to it:

[3] This is the same cable used on the Raspberry Pi for Composite output

[4] This is the same cable used on the Raspberry Pi for Composite output

[5] **IMPORTANT**: The dastaZ80DB RTC circuit does not work with rechargable batteries. Only use LIR2032 on the dastaz80 original.

- dastaZ80.img
- #

5. Introduce the SD card in the SD card slot at the back of the computer case. This procedure MUST be performed with the computer switched off. For the dastaZ80DB, the SD card slot is at the front of the case.
6. Connect the jack of the Audio/Video cable to the A/V connector at the back of the computer case. This procedure SHOULD be performed with the computer unplugged.
7. Connect the female power supply connector to the male connector at the back of the case.
8. For the dastaZ80 Original, connect the VGA cable from the monitor to the VGA connector at the back of the computer case. This procedure SHOULD be performed with the computer unplugged.
9. For the dastaZ80DB, there are two options:
   - Connect the TTL-to-USB (or TTL-to-RS-232) cable to the 4 pins header labelled *TTL I/O*, to have the input (keyboard) and output (screen) from/to another computer.
   - Or, connect the TTL-to-USB (or TTL-to-RS-232) cable to *TTL I/O*, but also connect the VGA cable to the VGA connector. You will then need to configure the Control Panel to use VGA output instead of TTL output. This way, the signal will be send to the VGA monitor and not to the other computer.

That's really it. Switch the Power Switch to the ON position and you should see text on your VGA monitor. The computer and DZOS are ready to use.

If you are using the dastaZ80DB with the TTL-to-USB (or TTL-to-RS-232) cable for both input and output (i.e. no VGA), you need to open a terminal program in your other computer and set it up to connect at 115,200bps 8N1 no hardware flow control. After flipping the switch, you should see text on the terminal program.

Nevertheless, it's worth noting that by following this procedure the disk image in the SD card will be empty, and therefore no software will be available. See how to add software to your disk image in the following subsection.

3.4 Transferring software to and from a disk image

3.4.1 Copy to a disk image

Software for the dastaZ80 can be created on a PC, or downloaded for example from the DZOS Software repository on Github[6]. But how do we transfer that software into a disk image on our SD card?

Another DZOS related repository is the Arduino Serial Multi-Device Controller for dastaZ80's dzOS, or ASMDC for short.

In this repository there is a tool, called *imgmngr* (Image Manager) which can be used to manipulate files inside disk images. It allows to add files, extract files, rename files, delete files, change the file attributes, and display the disk image contents.

Lets imagine we have a binary, called *helloworld*, in our PC, and we want to copy it to the disk image, called *dastaZ80.img*, that we created following the previous steps explained in the Setting up the system section. Simply execute *imgmngr* like: `imgmngr dastaZ80.img -add helloworld`

Now, if we introduce the SD card into the SD Card slot of the dastaZ80, switch it on and execute the DZOS command cat, we will see a file *helloworld*.

[6]DZOS Software repository: https://github.com/dasta400/dzSoftware

3.4.2 Copy from a disk image

What if we created a file in DZOS and we want to make a backup or simply share it with somebody else?

As we have seen, the same tool *imgmngr* can also be used to extract files from a disk image: `imgmngr dastaZ80.img -get helloworld`

Once finished, we will have a file *helloworld* in our PC, with the same contents as the file *helloworld* in the disk image.

3.5 Dual Video Output

The dastaZ80 has two simultaneous video outputs: VGA and Composite.

The VGA output, called *High Resolution*, is the default output for the Operating System. As it provides 80 columns, it is ideal for applications.

The Composite output, called *Low Resolution*, can only provide 40 columns in Text Mode and 32 in Graphics Modes[7], making it less ideal for applications. But in contrast to the VGA output, it offers graphics and hardware sprites, so it makes it more suitable for video games and graphics output from applications.

3.6 Dual Joystick Port

The **Dual Digital Joystick Port** consist of two DE-9 Male connectors for connection of one or two Atari joystick ports.

Compatible joysticks are those used on Atari 800, Atari VCS, Atari ST, VIC-20, C64, Amiga and ZX Spectrum. Other joysticks, like the ones for the Amstrad CPC and MSX, changed the +5V and GND pins, so it MUST not be used.

[7] This is a limitation imposed by the Video Display Controller used, a Texas Instruments TMS9918A.

4 Computer Operation

4.1 dastaZ80DB Front Panel

The dastaZ80DB has a front panel (located for easy access) with buttons, switches, indicator lights and a MicroSD slot. Also, in difference to the dastaZ80 Original, this computer has a LCD Display and some push buttons which form the *Control Panel*.

The front panel contains:

- ON/OFF Switch: use it to turn on and off the computer.
- Micro SD Card slot: for massive storage of data.
- Control Panel:
  - LCD display: for showing information.
  - Three push buttons (Up, Down, Select): for operating the Control Panel.
- Light (LED) indicators (from left to right):
  - Halted (purple): lighted when the system is in *Halt* status.
  - Bi-colour LED:
    - Reset (red): lighted when the system is performing a hardware reset.
    - ROM Paged (blue): lighted when the **ROM** chip has been disconnected. Hence, full **RAM** is available.
  - SD card (yellow): blinks when the SD Card is being accessed (reading or writing).
  - Power (green): lighted when the system is switched ON.

4.1.1 Control Panel

This Panel is used to configure some pre-booting configurations (if you are familiar with IBM AS/400[8] computers you will see where this idea comes from). It also monitors the internal temperature of the box and switches ON and OFF a small fan used for cooling.

The Panel consists of one LCD display and three push buttons. The buttons are labelled +, - and *Select*.

The LCD displays what is referred in this manual as *pages* of information. The buttons + and - are used to navigate the pages, and the button *Select* is used to select a specific configuration shown on a page.

The LCD display is always ON as soon as the computer has been plugged to the 5V/4A power supply. That's right, even when the computer is OFF, this panel is ON.

Once the computer is switched ON (via the Power Switch), the configuration cannot be changed. Hence, any selection via the *Select* button will be ignored.

Changes in configuration are not stored. Once the computer is unplugged from the power supply, it will start again with the default settings.

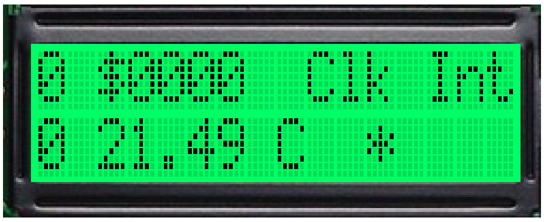
[8] The IBM AS/400 (Application System/400) is a family of midrange computers from IBM produced since 1988.

Current Configuration - Page 0

When plugged in, the display will show the Page 0. This is indicated by the number zero shown in the first character of each of the two lines of the display.

Page 0 shows the Current Configuration. If the computer was to be switched ON at this moment, it will use the configuration displayed in the LCD to its boot.

Page 0 information is:



- Line 1:
  - *0*: indicates that this is Page 0.
  - *$0000*: refers to the address in ROM that will be used to start reading the OS.
  - *Clk Int*: indicates the the Internal clock is being used as system clock.
- Line 2:
  - *0*: indicates that this is Page 0.
  - *21.49 C*: is the current measured internal temperature of the box.
  - ***: this only appears if a certain threshold of maximum temperatire has been reached. If the computer is switched ON, here it will appear a spinning animation indicating that the fan is ON. If the computer is OFF, a blinking exclamation mark will appear, indicating that it may not be safe to switch on the computer.

ROM Start Address - Pages 1 and 2

Pages 1 and 2 are selectable pages, and show the two different start ROM addresses that can be selected.

The EEPROM that contains the OS can have two different versions. One version is burned into the EEPROM at address `0x0000` and the other at `0x4000`. This is useful for testing new versions or even having completely different operating systems.

To select one or the other, press the button *Select* while in the desired page. After a couple of seconds, the Panel will display Page 0 automatically, and you should see the selected value in its corresponding position.

HardDisk Drive - Pages 3 and 4

Pages 3 and 4 are selectable pages, and show the two different HDD that can be selected.

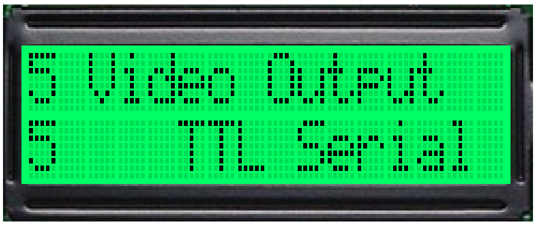
The two options are:

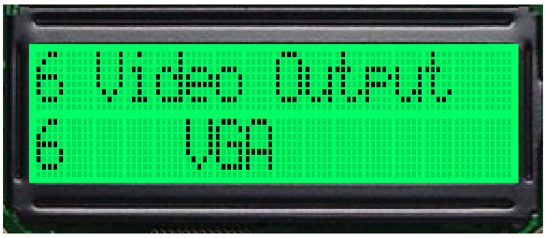
- **Internal HHD**: This is the Sd Card controlled by the **ASMDC**, and consist of a MiniSD card slot that allows the insertion/extraction of MiniSD cards containing Disk Image Files.
- **External HDD**: At the back of the computer, there is a 6-pin header to which an FTDI-to-USB cable can be connected. On a PC, connected to the USB end of the cable, a program (called *SerialHDDsimul*) can be run and simulate the behaviour of the **ASMDC**, but with 3.8 times faster access.

Video Output - Pages 5 and 6

Pages 5 and 6 are selectable pages, and show the two different video output that can be selected.

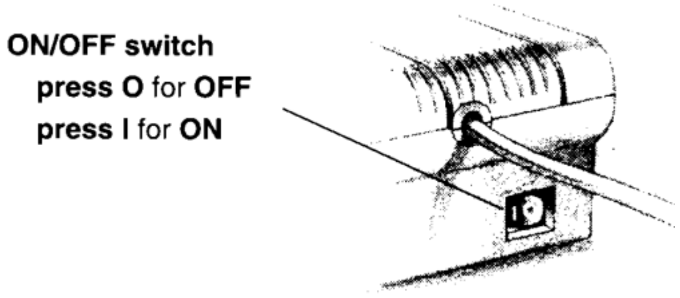
You MUST change this configuration accordingly to how you did set up the computer (as explained in the section Setting up the system). Select *TTL Serial* if you are using a TTL-to-USB (or TTL-to-RS-232) cable for input and output, or Select *VGA* if you are using TTL-to-USB (or TTL-to-RS-232) cable for input but VGA for output.





4.2 ON/OFF button

The ON/OFF button is located on the lefthand side of the back of the dastaZ80 computer and on the front of the dastaZ80DB.



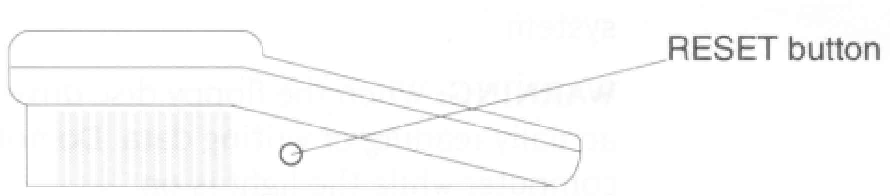
This button is used to turn ON and OFF the computer.

Before turning it ON, read the instriuctions on the section Setting up the system of this manual.

Before turning it OFF, it is highly recommended to use the command halt to ensure that all **DISK** data has been correctly saved. Otherwise, corruption of data may occur.

4.3 Reset button

The reset button is located on the lefthand side on both the dastaZ80 and the dastaZ80DB.



This button is used to restart the computer without turning it off via the ON/OFF button.

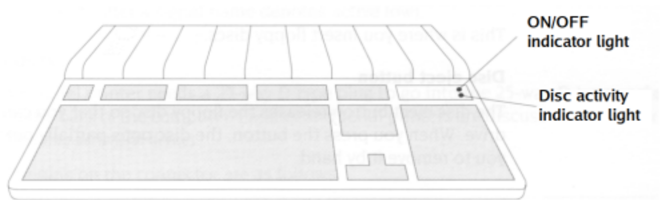
To reset the computer simply press and release the button. The reset process will take 6.5 seconds in total, as explained in the section *Reset circuit* of the dastaZ80 Technical Reference Manual[2].

4.4 Indicator LEDs

Indicator LEDs for the dastaZ80DB has been discussed in previous section dastaZ80DB Front Panel.

For the dastaZ80, at the top of the computer case there a few labelled LEDs that give information of the status of several internal parts of the computer.

- Above the numeric pad, on the righthand side of the keyboard, there are two LEDs:
  - **POWER**. This LED is always on when the computer is switched on via the ON/OFF button. It glows in orange colour.
  - **DISC**. This LED blinks whenever a **DISK** operation (read/write) is happening. It glows in green colour.



- Above the *Esc* and function keys (*F1-F12*), on the lefthand side of the keyboard, there are two LEDs. These LEDs are multi-colour, hence glowing at different colour each:
  - Computer status
    - **RESET**. Lighted in red colour when the computer is in reset status. This happens for 6.5 seconds when the computer is switched ON (via the ON/OFF button) and when the computer is reset via the Reset button.
    - **HALTED**. Lighted in purple when the computer is in halt status. Usually after issuing the command halt.
    - **ROM PAGED**. Lighted in blue when the **ROM** has been electrically disconnected and therefore the computer will only perform operations (read/write) from/to the **RAM**. This happens only during the Boot Sequence. See the section *OS Boot Sequence* of the dastaZ80 Programmer's Reference Guide[7] for more detailed information about the Boot Sequence.
  - Clock Selection
    - **CLOCKSEL**. It glows in yellow colour when the Internal clock is used. And in purple colour when an External clock is being used by the computer.

4.5 MicroSD

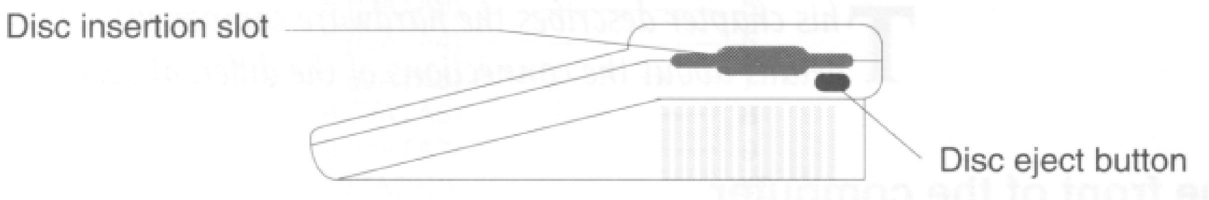
At the back of the dastaZ80 computer and on the front of the dastaZ80DB there is a MicroSD slot.

Insert here a MicroSD formatted with FAT32 and containing Disk Image Files formatted with DZFS[9].

4.6 3.5 inch Floppy Disc Drive

The Floppy Disc Drive is located on the righthand side of the dastaZ80 computer and on the front of the dastaZ80DB.

3.5 inch floppy discs can be inserted in this drive.



To remove an inserted floppy disc from the drive, press the *Disc eject button*. The disc will partially pop out, allowing you to completelly remove it by hand.

[9] DZFS (dastaZ80 File System) is a file system of my own design, for mass storage devices, aimed at simplicity

4.7 Attaching peripheral devices

4.7.1 TTL I/O

The dastaZ80DB has a 6-pin header labelled *TTL I/O* that is used for connecting either a TTL-to-USB cable or a TTL-to-RS232 converter.

This connector carries the signals for the input (keyboard) and output (screen).

By connecting the dastaZ80DB to another computer, the latter can *control* the former. As a matter of fact, there is no other way to input commands to the dastaZ80DB than using another device (a computer in most cases, but also a serial terminal can be used).

For the output, two options are available:

- *TTL*: output to a serial device (usually another computer running a terminal program).
- *VGA*: output to a VGA monitor.

The option *VGA* MUST be selected in the page 6 (Video Output) of the Control Panel).

The TTL connector pins are configured as follows (seeing it from the front):

Ground — NC — NC — **TX** — **RX** — NC

NC, stands for Not Connected. These are the *+5V*, */CTS* and */RTS* signals, which are not used.

4.7.2 Dual Video Output

At the back of the computer you will find two connectors, beside each other, for the connection of video output.

The first one, and necessary to use the computer, is a VGA connector for the **VGA video output**.

Plug here a standard VGA cable connected to a VGA monitor.

The second connector, which is optional (i.e. the computer will function perfectly normal without this connected), is a 3.5mm female jack for the NTSC **Composite video output**.

Plug here the jack side of a 3.5mm jack-to-3-RCA Audio/Video cable [10].

[10] This is the same cable used on the Raspberry Pi for Composite output.

Connect the yellow RCA cable of the jack-to-3-RCA cable to the Composite input of a monitor or TV.

4.7.3 Stereo Sound Output

The stereo sound signal comes out of the same connector used for the Composite video output.

Connect the jack-to-3-RCA white and red cables to a pair of speakers or to the input of a sound system amplifier.

4.7.4 USB Keyboard for External computer

The keyboard of the dastaZ80 can be used as an USB keyboard on other computers.

Connect the a USB cable between this connector and your other computer, switch on dastaZ80 and press the key *ScrollLock*. From now on (as indicated by the ScrollLock LED being lighted) dastaZ80 will not read any keystrokes, but instead will send them to the computer connected via USB.

If you want to use dastaZ80 at any time, just press *ScrollLock* again. There is no need to unplug the USB cable. The *ScrollLock* key is doing the switching.

This feature can be handy when you are using dastaZ80 and another PC at the same time and don't want to be switching hands between two keyboards all the time. It saves space too!

4.7.5 ROM Cartridge

Refer to the section *Cartridge Port* of the dastaZ80 Technical Reference Manual[2] for more detailed information.

4.7.6 External HDD

The dastaZ80DB has a 6-pin header labelled *External HDD* that is used for connecting either a TTL-to-USB cable or a TTL-to-RS232 converter.

This connector carries the signals for the Transmit (*TX*) and receive (*RX*) of an external **DISK** device.

By connecting the dastaZ80DB to another computer, the latter can provide a fast hard disk drive simulation (3.8 times faster than the SD Card on the **ASMDC**).

The option *External* MUST be selected in the page 4 (HardDisk Drive) of the Control Panel).

The TTL connector pins are configured as follows (seeing it from the front):

Ground — NC — NC — **TX** — **RX** — NC

NC, stands for Not Connected. These are the *+5V*, */CTS* and */RTS* signals, which are not used.

4.8 Developer Mode

At the back of the dastaZ80 is located a pin header that configures the routing of the signals between the the serial device and the internal I/O devices.

By default, the signals are routed as follows:

- Serial TX → VGA controller RX

- Serial RX ← Keyboard controller TX

It's possible to redirect these signals to an external device (e.g. a computer running a terminal emulator software like Minicom or PuTTY). This configuration allows to use the computer via an external keyboard and/or outputting the video signal to that device.

In the dastaZ80DB, this pin header is the TTL I/O, and signals are not routed because is used for connecting either a TTL-to-USB cable or a TTL-to-RS-232 converter.

It's called *Developer Mode*, because with this configuration is very easy (and quick), to send programs via the pastefile tool and test them. Basically, we can write programs directly to the **RAM**, without the need of transfer them to the MicroSD card.

5 Computer Maintenance

5.1 Internal Battery

While it is unplugged from the main power or switched off, the computer keeps the date and time, thanks to its Real-Time Clock (RTC) internal circuitry.

This RTC is supported by a LIR2032 Li-Ion rechargeable 3.6V button cell battery in the dastaz80 original, and a CR2016 non-rechargeable 3V button cell battery in the dastaZ80DB.

For the LIR2032, to maintain the battery at full charge, ideally the computer SHOULD be switched ON for at least one hour every month. Nevertheless, after a few years, due the way rechargeable batteries work, the battery may be unable to recharge anymore. In such case, the baterry MUST be replaced with a new battery of the same characteristics.

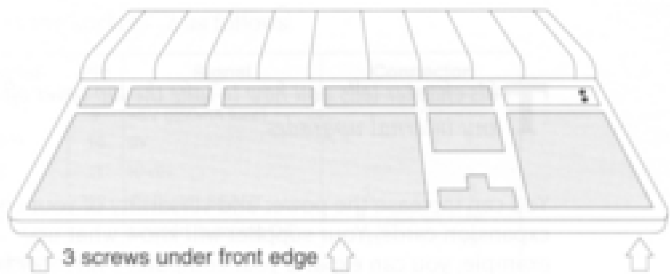
In the case of the CR2016, the battery charge will be consumed after a certain number of hours of use, independently of how often or long the computer is powered ON.

In any case, if you are not planning to use the computer for a year or more, it is highly recommended to remove the battery, to avoid leaking of chemical fluids that are highly toxic and also can damage the internal circuitry.

There is no danger of electrical shock, as the computer is powered just by 12V max., but it is highly recommended to unplug the computer to avoid possible short circuits that could damage the internal circuitry. It is also recommened to discharge yourself from static electricity before touching the inside of the computer.

5.1.1 dastaZ80 Original

To replace/remove the battery you need to open the computer case, by unscrewing the three small cross-head screws under the front edge of the computer case.



You can use either a rechargeable LIR2032 battery or a non-rechargeable CR2016.

5.1.2 dastaZ80DB

To replace/remove the battery you need to open the computer case, by lifting the cover from the left side of the computer.

You MUST use only a non-rechargeable CR2016 battery.

5.2 Environment and Cleaning the computer

In general, avoid high humidity, extreme cold, and extreme heat environments... and do not put the computer in the dishwasher!

For the dastaZ80, it is highly recommended to use a cover to avoid the concentration of dust in the keyboard, which can lead to false contacts.

The dastaZ80DB has an internal fan that will blow air to the outside when a certain temperature inside the box has been reached. To ensure a good flow of air, ensure there is a good gap between the back of the computer and any object behind the computer.

Before switching ON the dastaZ80DB, if you see a blinking exclamation mark on the Page 0 (as explained in section Control Panel) DO NOT switch the computer ON until it cooled down.

6 Operating System (OS)

dzOS (or DZOS) is a single-user single-task ROM-based operating system (OS) for the 8-bit homebrew computer dastaZ80. It is heavily influenced by ideas and paradigms coming from Digital Research, Inc. CP/M, so some concepts may sound familiar to those who had used this operating system.

The user communicates with the OS via a keyboard and a screen connected directly to the computer.

The main job of dzOS is to allow the user to run programs, one at a time and communicate with the different peripherals (or devices, as referred in this manual). The user types in a command and the operating system checks what to do with the command received, to execute a set of instructions.

Other tasks of dzOS are: handling disk files via its file system (DZFS), getting user input from the keyboard, writing messages on the screen and receiving/sending data through the serial port.

dzOS consists of three parts:

- The **BIOS**, that provides functions for controlling the hardware.
- The **Kernel**, which provides general functions for everything that is not hardware dependent.
- The Command-Line Interface (**CLI**), that provides commands for the user to talk to the Kernel and the BIOS.

The Kernel and the CLI are hardware independent and will work on other Z80 based computers. Therefore, by adapting the BIOS code, dzOS can easily be ported to other Z80 systems.

6.1 dastaZ80 File System (DZFS)

A file system manages access to the data stored in a storage medium, a MicroSD card or a Floppy Disk in the case of dastaZ80, and allows the OS to load and save data in the device (from now on, referred as **DISK**).

DZFS is my first time designing a file system and for this reason I kept it very simple.

It uses *Logical Block Addressing* (LBA) for accessing the data on the **DISK**, and an allocation table called *Block Allocation Table* (or *BAT* for short) based in blocks of sectors.

A **DISK** in DZFS can be a maximum of 33,521,152 bytes (33 MB). The reason for this specific maximum is explained later in this section. The Z80 is a 16-bit addressing CPU, and hence it can only access a maximum of 65,536 addresses. Therefore, it would be impossible for the CPU to access bytes with a higher address. To solve this, the data in the **DISK** is virtually grouped into Sectors. Each Sector is a group of 512 bytes. Therefore, we have 65,472 Sectors (33,521,664 / 512) per disk, which is addressable by the CPU.

For this reason, the BAT is really not allocating Blocks but Sectors, it should instead have been more correctly named *Sector Allocation Table.*

As the free RAM of dastaZ80 is about 48 KB, it makes no sense to have files bigger than that, as it would not fit into **MEMORY**. Therefore, I have decided that each Block can store only a single file.

The file index is kept in an allocation table that occupies an entire Block (32,768 bytes) and each entry in the table is 32 bytes long. Therefore, the *BAT* can allocate a maximum of 1,024 files (32,768 / 32).

Because there is a maximum of 1,024 files, and each file can be a maximum of 32,768 bytes, the maximum amount of bytes that can be stored in a **DISK** is 33,554,432 bytes (32,768 x 1,024). The first 512 bytes are used by the *Superblock* (to store **DISK** geometry and other details), and the BAT uses 32,768 bytes in itself. Therefore, there is a maximum space left for storing files of 33,521,152 bytes.

Each entry in the *BAT* holds information for each file in the disk; filename, attributes, time/date created, time/date last modified, file size, load address.

6.1.1 File Attributes

Files can have any of the following attributes:

- **Read Only** (R): it cannot be overwritten, renamed or deleted.
- **Hidden** (H): it does not show up in the results produced by the command cat.
- **System** (S): this is a file used by DZOS and it MUST not be altered.
- **Executable** (E): this is an executable file and can be run directly with the command run.

6.1.2 File Types

- **USR**: User defined.
- **EXE**: Executable binary. Meant to be run with the run command.
- **BIN**: Binary (non-executable) data. Meant to be loaded into **MEMORY** with the load command.
- **BAS**: BASIC code. Meant to be laoded with MS BASIC.
- **TXT**: Plain ASCII Text file.
- **FN6**: Font (6×8) for Text Mode. Meant to be loaded with the loadfont tool.
- **FN8**: Font (8×8) for Graphics Modes. Meant to be loaded with the loadfont tool.
- **SC1**: Screen 1 (Graphics I Mode) Picture. Meant to be loaded with the loadscr tool.
- **SC2**: Screen 2 (Graphics II Mode) Picture. Meant to be loaded with the loadscr tool.
- **SC3**: Screen 3 (Multicolour Mode) Picture. Meant to be loaded with the loadscr tool.

6.1.3 DZFS limitations

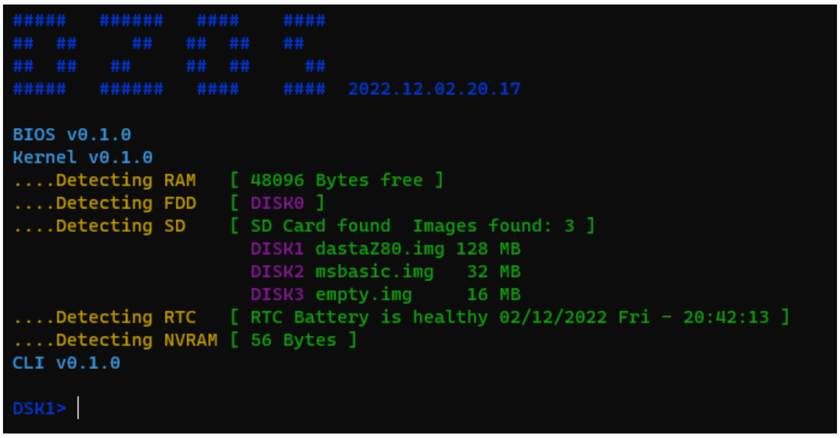
The current version of the DZFS implementation (DZFSV1) have the following limitations:

- No support for directories. All files are presented at the same level.
- Filenames:
  - Are case sensitive.
  - Can be maximum 14 characters long.
  - Can only contain alphabetical (A to Z) and numerical (0 to 9) letters.
  - Cannot start with a number.
  - No support for extensions. But it uses attribute File Type instead.
- Maximum size for a file is 32,768 bytes.

6.2 The Command Prompt

When you switch ON the computer, you will hear a low tone (beep).

The **Low Resolution Display** will display a welcome text and the **High Resolution Display** will show some information:



This information tells you about the release version of DZOS (2022.07.19.13 in the screenshot). The BIOS, Kernel and CLI versions, and the detection of the different devices used by the computer. It also tells about whichs **DISK**s are available.

After that information,you will see the *command prompt*. It starts with the letters *DSK* (short for DISK) and a number, followed by the symbol >

The number indicates which **DISK** is currently used for **DISK** operations.

In other words, if you see *DSK0*, it means that the Floppy Disk Drive (**FDD**) is selected. Entering commands like *cat*, *diskinfo*, *load*, etc., will instruct the computer to do it on the **FDD**.

7 OS Commands

There are a number of commands included in the operating system. These commands are stored in **MEMORY** at boot time, and therefore can be called at any time from the command prompt.

Some commands may have mandatory and/or optional parameters. These parameters MUST be entered in the order listed. Interchanging the order of parameters will result on undesired behaviour.

Parameters can be separated either by a comma or a space. For clarity, in this document all parameters are separated by a comma.

Programs stored in **DISK** can be executed directly by simply entering the filename as a command. But only those in the current **DISK** (see command dsk) and with attribute *EXE*.

7.1 General Commands

7.1.1 peek

Prints the value of the byte stored at a specified **MEMORY** address.

> **peek <*address*>**

Parameters:

address: address where the user wants to get the value from.

Example: > `peek 41A0`

Will print (in hexadecimal) whatever byte is at location 0x41A0.

7.1.2 poke

Changes the value of the byte stored at a specified **MEMORY** address.

> **poke <*address*>,<*value*>**

Parameters:

address: address where the user wants to change a value.

value: new value (in Hexadecimal notation) to be stored at *address*.

Example: > `poke 41A0,2D`

Will overwrite the contents of the address 0x41A0 with the value 0x2D.

7.1.3 autopoke

Allows the user to enter a series of values to be stored at the starting address and its consecutive addresses. Think of it like a way to do poke but without having to enter the **MEMORY** address each time.

After entering the command, a different command prompt, denoted by the symbol $, will be displayed.

Values are entered one by one after the symbol $. Pressing *Return* with no value will end the command.

> **autopoke <*address*>**

Parameters:

address: address where the user wants to start changing values.

Example: > `autopoke 41A0`

Will overwrite the contents of the address 0x41A0 with the first value entered by the user at the $ prompt. Next value entered will overwrite the contents of the address 0x41A1, next 0x41A2, and so on until the end of the command.

7.1.4 halt

Tells the **DISK** controller to close all files, disables interrupts and puts the CPU in halted state, effectively making the computer unusable until next power cycle (*Have you tried turning it off and on again?*).

SHOULD be used before switching the computer off, to ensure all **DISK** data has been correctly saved. MUST not be used while the busy light of the **DISK** is on.

> **halt**

Parameters: None

7.1.5 run

Transfers the Program Counter (PC) of the Z80 to the specified address. In other words, this command is used to directly run code that has been already loaded in **RAM**, for example with the command load.

> **run** <***address***>

Parameters:

address: address from where to start running.

Example: > `run 4420`

The **CPU** will start running whatever instructions finds from `0x4420` and onwards. Programs run this way MUST end with a jump instruction (JP) to CLI prompt address, as described in the *dastaZ80 Programmer's Reference Guide*[7]. Otherwise the user will have to reset the computer to get back to CLI. Not harmful but cumbersome.

7.1.6 crc16

Generates a CRC-16/BUYPASS1[11]

There are two formats of this command:

crc16 <***start_address***> <***end_address***> and **crc16** <***filename***>

Here the former is described. See the section *7.3 DISK Commands* on page 22 for the other format.

> **crc16** <***start_address***> <***end_address***>

Parameters:

start_address: first address from where the bytes to calculate the CRC will be read.

end_address: last address from where the bytes to calculate the CRC will be read.

Example: > `crc16 0000,0100`

Will calculate the CRC of all bytes in MEMORY between the two specified address and show it on the screen:

```
CRC16: 0x2F25
```

[11]A 16-bit cyclic redundancy check (CRC) based on the IBM Binary Synchronous Communications protocol[8] (BSC or Bisync). It uses the polynomial $X^{16} + X^{15} + X^2 + 1$

7.1.7 clrram

Fills with zeros the entire Free RAM area (i.e. from 0x4420 to 0xFFFF).

> **clrram**

Parameters: None

7.2 Real-Time Clock (RTC) Commands

7.2.1 date

Shows the current date and day of the week from the Real-Time Clock (**RTC**).

> **date**

Parameters: None

Will show (will differe depending on the date on the **RTC**):

```
Today: 22/11/2022 Tue
```

7.2.2 time

Shows the current time from the Real-Time Clock (**RTC**).

> **time**

Parameters: None

Will show (will differe depending on the time on the **RTC**):

```
Now: 16:24:36
```

7.2.3 setdate

Changes the current date stored in the Real-Time Clock (**RTC**).

> **setdate <*yy*><*mm*><*dd*><*dow*>**

Parameters:

yy: year.

mm: month.

dd: day.

dow: day of the Week. (1=Sunday).

Example: > `setdate 2211032`

7.2.4 settime

Changes the current time stored in the Real-Time Clock (**RTC**).

> **settime <*hh*><*mm*><*ss*>**

Parameters:

hh: hour.

mm: minutes.

ss: seconds.

Example: > `settime 185700`

7.3 Disk Commands

7.3.1 cat

Shows a catalogue of the files stored in the **DISK**.

> **cat**

Parameters: None

Example: > `cat`

Will show (will differ depending on the contents of your **DISK**):

```
Disk Catalogue
----------------------------------------------------------------------
File        Type  Last Modified        Load Address  Attributes  Size
----------------------------------------------------------------------
HelloWorld  EXE   12-03-2022 13:21:44  4420          R SE        38
file2       TXT   11-05-2022 17:12:45  0000          SE          241
```

By default, deleted files are not shown in the catalogue. To show also deleted files do a *poke 40ac, 01*. And a *poke 40ac, 00* to hide them again.

Deleted files are identified by a ˜symbol in the first character of the filename.

```
Disk Catalogue
----------------------------------------------------------------------
File        Type  Last Modified        Load Address  Attributes  Size
----------------------------------------------------------------------
~elloWorld  EXE   12-03-2022 13:21:44  4420          R SE        38
file2       TXT   11-05-2022 17:12:45  0000          SE          241
```

7.3.2 erasedsk

Overwrittes all bytes of all sectors in a **DISK** in the **FDD**, with `0xF6`

<u>This is a destructive action</u> and it makes the **DISK** unusable to any (included dzOS) computer, as there is no file system in the disk after the command is completed.

Before it can be used by dzOS, the command *formatdsk* MUST be executed.

It is recommended to only use this command in the case of wanting to destroy all data in a **DISK**, because *formatdsk* doesn't actually delete any data, or to check if a Floppy Disk is faulty. Otherwise, the command *formatdsk* SHOULD be the right command for normal usage of the computer.

> **erasedsk**

Parameters: None

Example: > `erasedsk`

7.3.3 formatdsk

Formats a **DISK** with DZFS format. <u>This is a destructive action</u> and makes the **DISK** unsuable by any computers not using DZFS as their file system. It overwrites the DZFS *Superblock* and *BAT*.

> **formatdsk <*label*>**

Parameters:

label: a name given to the **DISK**. Useful for identifying different disks. It can contain any characters, with a maximum of 16.

Example: > `formatdsk mainDisk`

Will format the SD card inserted in the SD card slot at the back of the computer case, having *mainDisk* as disk label.

7.3.4 load

Loads a file from **DISK** to **RAM**.

The file will be loaded in **RAM** at the address from which it was originally saved. This address is stored in the DZFS BAT and cannot be changed.

> **load <*filename*>**

Parameters:

filename: the name of the file that is to be loaded.

Example: > `load HelloWorld`

Will load the contents (bytes) of the file *HelloWorld* and copy them into the **RAM** address from which it was originally saved.

7.3.5 rename

Changes the name of a file.

> **rename <*current_filename*>, <*new_filename*>**

Parameters:

current_filename: the name of the file as existing in the **DISK** at the moment of executing this command.

new_filename: the name that the file will have after the command is executed.

Example: > `rename HelloWorld,Hello`

Will change the name of the file *HelloWorld* to *Hello*.

7.3.6 delete

Deletes a file from the **DISK**.

Technically is not deleting anything but just changing the first character of the filename to a ˜symbol, which makes it to not show up with the command *cat*. Hence, it can be undeleted by simply renaming the file. But be aware, when saving new files DZFS looks for a free space [12] on the **DISK**, but if it does not find any it starts re-using space from files marked as deleted and hence overwriting data on the **DISK**.

> **delete <*filename*>**

Parameters:

filename: the name of the file that is to be deleted.

[12] By free space on the **DISK** we understand a Block in the DZFS BAT that was never used before by a file.

Example: > `delete HelloWorld`

Will delete the file *HelloWorld*.

7.3.7 chgattr

Changes the File Attributes of a file.

> **chgattr <*filename*>,<*RHSE*>**

Parameters:

filename: the name of the file to change the attributes.

RHSE: the new attributes (see list above) that are to be set to the specified file. Attributes are actually not changed but re-assigned. For example, if you have a file with attribute *R* and specified only *E*, it will change from Read Only to Executable. In order to keep both, you MUST specify both values, *RE*.

Example: > `chgattr HelloWorld,RE`

Will set the attributes of the the file *HelloWorld* to Read Only and Executable.

7.3.8 save

Saves the bytes of specified **MEMORY** addresses to a new file in the **DISK**.

> **save <*start_address*>, <*number_of_bytes*>**

Parameters:

<***start_address***>: first address where the bytes that the user wants to save are located in **MEMORY**.

<***number_of_bytes***>: total number of bytes, starting at *start_address* that will be saved to **DISK**.

Example: > `save 4420,512`

Will create a new file, with the name entered by the user when prompted, with 512 bytes of the contents of **MEMORY** from 0x4420 to 0x461F.

7.3.9 dsk

Changes current disk for all **DISK** operations.

> **dsk <*n*>**

Parameters:

<***n***>: **DISK** number.

Example: > `dsk 0`

Will change to **FDD**, and all the **DISK** operations will be performed in the **FDD** until the next boot or a new *dsk* command.

The CLI prompt changes to indicate which disk is in use.

7.3.10 diskinfo

Shows some information about the **DISK**.

> **diskinfo**

Parameters: None

Example: > `diskinfo`

Will show (will differ depending on the contents of your **DISK**):

```
Disk Information
        Volume . . :  dastaZ80 Main        (S/N: 352A15F2)
        File System:  DZFSV1
        Created on :  03/10/2022 14:22:32
        Partitions :  01
   Bytes per Sector:  512
  Sectors per Block:  64
```

7.3.11 disklist

Shows a list of all available **DISK** (**FDD** and Disk Image Files on the **SD**).

> **> disklist**

Parameters: None

Example: > `disklist`

Will show (will differ depending on the Disk Image Files on your **DISK**):

```
DISK0  FDD
DISK1  dastaZ80.img  128 MB
DISK2  msbasic.img    32 MB
DISK3  empty.img      16 MB
```

IMPORTANT: When the list (210 bytes in total, for a maximum of 15 Disk Image Files) is retrieved from the **ASMDC**, dzOS stores it at the very bottom of the RAM (`0xFF2D`). In case that you may have a program loaded that uses those low bytes, after executing the *disklist* command the program will be corrupted.

7.4 VDP (*Low Resolution Screen*) Commands

7.4.1 vpoke

Changes the value of the byte stored at a specified **VRAM** address.

> **> vpoke <*address*>,<*value*>**

Parameters:

> ***address***: address where the user wants to change a value.
>
> ***value***: new value (in Hexadecimal notation) to be stored at *address*.

Example: > `vpoke 3800,00`

Will overwrite the contents of the address 0x3800 of the **VRAM** with the value 0x00.

7.4.2 screen

Changes the **Low Resolution Screen** display mode.

> **> screen <*mode*>**

Parameters:

> ***mode***: one of the valid **Low Resolution** Screen Modes:

- **0**: **Text Mode**.
- **1**: **Graphics I Mode**.
- **2**: **Graphics II Mode**.
- **3**: **Multicolour Mode**.
- **4**: **Graphics II Mode Bitmapped**.

Example: > `screen 0`

Will put the **Low Resolution Screen** in Text Mode.

7.4.3 clsvdp

Clears the contents of the **Low Resolution Screen**.

> **clsvdp**

Parameters: None

8 Other Software

8.1 Memory Dump (memdump)

This program shows the contents (bytes) of a specified range of **MEMORY**.

The contents are printed as hexadecimal bytes, in groups of 16 per each line and with the printable ASCII value (if printable) or just a dot (if not printable).

At the start of the program, the user will be asked to enter the *Start Address* and the *End Address*. In the case of leaving blank (i.e. just press the *Return* key without entering any value), the program will terminate.

Example for *Start Address* = 0B40 and *End Address* = 0BEF:

```
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0B40:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00  ................
0B50:  21 3A 0F CD BE 03 06 01 CD 20 04 CD 4D 0C 21 56  !:...... ..M.!V
0B60:  0F CD BE 03 21 C4 22 3E 00 32 C4 22 CD 75 0B CD  ....!.">.2.".u..
0B70:  B1 0B C3 5B 0B CD C8 03 FE 20 CA 91 0B FE 2C CA  ...[..... ....,.
0B80:  91 0B FE 0D CA B0 0B 77 23 C3 75 0B C9 2B C3 75  .......w#.u..+.u
0B90:  0B 3A E4 22 FE 00 CA A4 0B 3A 04 23 FE 00 CA AA  .:.".....:.#....
0BA0:  0B C3 75 0B 21 E4 22 C3 75 0B 21 04 23 C3 75 0B  ..u.!.".u.!.#.u.
0BB0:  C9 21 C4 22 7E FE 00 CA 5B 0B 11 29 14 CD 02 0C  .!." ...[..)....
0BC0:  CA 89 0E 11 10 14 CD 02 0C CA 93 0E 11 2C 14 CD  .............,..
0BD0:  02 0C CA 55 0E 11 25 14 CD 02 0C CA 15 0F 11 15  ...U..%.........
0BE0:  14 CD 02 0C CA 9A 0E 11 1A 14 CD 02 0C CA C7 0E  ................
```

If the information reaches the bottom of the screen, a message will be shown to let the user decide what to do next:

```
[SPACE] for more or another key to stop
```

8.2 Video Memory Dump (vramdump)

This program shows the contents (bytes) of a specified range of **VRAM**.

The contents are printed as hexadecimal bytes, in groups of 16 per each line.

At the start of the program, the user will be asked to enter the *Start Address* and the *End Address*. In the case of leaving blank (i.e. just press the *Return* key without entering any value), the program will terminate.

Example for *Start Address* = 0000 and *End Address* = 00AF:

```
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000:  00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF
0010:  00 00 00 00 00 01 03 07 00 00 00 00 00 80 C0 E0
0020:  07 03 01 00 00 00 00 00 E0 C0 80 00 00 00 00 00
0030:  F0 F0 F0 F0 00 00 00 00 F0 F8 FC FE FF FF FF FF
0040:  0F 1F 3F 7F FF FF FF FF FF FF FF FF FE FC F8 F0
0050:  FF FF FF FF 7F 3F 1F 0F 00 00 00 00 00 00 00 00
0060:  0B C3 75 0B 21 E4 22 C3 75 0B 21 04 23 C3 75 0B
0070:  C9 21 C4 22 7E FE 00 CA 5B 0B 11 29 14 CD 02 0C
0080:  CA 89 0E 11 10 14 CD 02 0C CA 93 0E 11 2C 14 CD
0090:  02 0C CA 55 0E 11 25 14 CD 02 0C CA 15 0F 11 15
00A0:  14 CD 02 0C CA 9A 0E 11 1A 14 CD 02 0C CA C7 0E
```

If the information reaches the bottom of the screen, a message will be shown to let the user decide what to do next:

```
[SPACE] for more or another key to stop
```

8.3 Load Screen dumps (loadscr)

This program loads screen dumps, saved as raw data, to the the VRAM. It is in essence a picture display program.

In Mode 2 (Graphics II Mode bitmapped), screen data dumps are files of 14,336 bytes in length, composed by:

- Dump of the Pattern Table (6,144 bytes)
- Dump of the Sprite Pattern Table (2,048 bytes) filled with zeros
- Dump of the Colour Table (6,144 bytes)

In dzOS, these files are identified as File Type *SC1* (Graphics I Mode), *SC2* (Graphics II Bitmapped Mode) and *SC3* (Multicolour Mode).

8.4 Load Font (loadfont)

This program loads font files, which contain pattern definitions for text characters to be used for text display.

Mode 0 (Text Mode) uses 6x8 bytes characters. The rest of the modes use 8x8 bytes characters.

In dzOS, these files are identified as File Type *FN6* and *FN8* respectively.

8.5 MS BASIC 4.7b

The Nascom 2 computer[13] came with MS BASIC 7.4 installed in ROM, and the disassembled code was published in the *80-BUS NEWS* magazine [9], [10], [11], [12], [13], [14], [15].

Grant Searle published a modification in his Grant's *7-chip Z80 computer* webpage[16].

Grant's version was then modified to run on dastaZ80 under dzOS, adding commands like *LOAD*, *SAVE*, *VPOKE*, *VPEEK*, and more.

8.5.1 MS BASIC characteristics

- Commands
  - There is no support for *ELSE* in *IF... THEN*. Instead, it MUST be done with another *IF... THEN*.
- Variables
  - The first character of a variable name MUST be a letter.
  - No reserved words may appear as as variable names.
  - Can be of any length,, but any alphanumeric characters after the first two are ignored. Therefore *COURSE*, *COLOUR* and*COMIC* are the same variable.
  - Integer numbers are signed (i.e. from -32,768 to +32,767). To refer to a location n above 32,767, you must provide the 2's complement number (i.e n-65536)
  - Flotaing point is in the range 1.70141E38 to 2.9387E-38

8.5.2 Speeding up programs

- Delete *REM* statements.
- Delete spaces. For example, *10FORA=0TO10* is faster than *10 FOR A=0 TO 10*
- Use *NEXT* without the index variable.
- Use variables instead of constants, especially in *FOR* loop and other code that is executed repeatedly.

[13] The Nascom 2 was a single-board computer kit issued in the United Kingdom in December 1979.

- Reuse variable names and keep the list of variables as short as possible. Variables are set up in a table in the order of their first appearance in the program. Later in the program, BASIC searches the table for the variable at each reference. Variables at the head of the table take less time to search.
- MS BASIC uses a *garbage collector* to clear out unwanted space. The frequency of grabage collection is inversely proportional to the amount of string space. The time garbage collection takes is proportional to the square of the number of string variables. To minimise the time, make string space as large as possible and use as few string variables as possible.

8.6 MS BASIC 4.7b Standard version

Standard version is referring to the version (*b*) adapted by Grant Searle [17] of the original NASCOM 2 version (4.7).

For more detailed information on commands, statements, functions, etc., refer to the Nascom 2 Microcomputer BASIC Programming Manuals[5].

8.6.1 Intrinsic Functions

- **ABS**: Returns absolute value of a number.
- **ASC**: Returns the ASCII code of the first character of a string.
- **ATN**: Returns arctangent in radians.
- **CHR$**: Returns a string whose one element has its ASCII code.
- **COS**: Returns cosine in radians.
- **EXP**: Returns the mathematical constant e (Euler's number) to the power of a specified number.
- **FRE**: Returns number of bytes in memory not being used by BASIC.
- **INP**: Reads a byte from a port.
- **INT**: Returns the largest integer of a floating number.
- **LEFT$**: Returns x leftmost characters of a string.
- **LEN**: Returns length of a string.
- **LOG**: Returns natural log of a number.
- **MID$**: Returns x characters of a string.
- **POS**: Returns present column position of terminal's print head..
- **RIGHT$**: Returns x rightmost characters of a string.
- **RND**: Returns a random number between 0 and 1.
- **SGN**: Returns 0 or 1 depending on the sign of a number.
- **SIN**: Returns the sine in radians.
- **SPC**: Prints x number of blanks.
- **SQR**: Returns square root of a number.
- **STR$**: Returns string representation of value of a number.
- **TAB**: Spaces to x position on the terminal.
- **TAN**: Returns tangent in radians.

- **USR**: Calls a user's machine language subroutine.
- **VAL**: Returns numerical value of a string.

8.6.2 Statements

- **DATA**, **DEEK**, **DEF**, **DIM**, **DOKE**, **END**, **FN**, **FOR...NEXT...STEP**, **GOTO**, **GOSUB**, **IF...THEN**, **INPUT**, **LET**, **LINES**, **ON...GOTO**, **ON...GOSUB**, **OUT**, **PEEK**, **POKE**, **PRINT**, **READ**, **REM**, **RESTORE**, **RETURN**, **STOP**, **WAIT**, ~~**WIDTH**~~[14]

8.6.3 Commands

- **CLEAR**: Sets all program variables to zero.
- **CLS**: Clears the screen.
- **CONT**: Continues program execution after *Escape* key was pressed, or a *STOP* or *END* statement has been executed.
- **LIST**: List the contents of the BASIC program in memory.
- **MONITOR**: Transfer command to dzOS Command-Line Interface (CLI).
- **NEW**: Deletes the current program and clears all variables.
- **NULL**: Sets the number of nulls to be printed at the end of each line.
- **RUN**: Starts execution of the current program.

8.6.4 Operators

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- **ˆ** Power of

8.6.5 Relational Operators

- >, <, <>, =, <=, >=

8.6.6 Logical Operators

- **AND**, **NOT**, **OR**

8.6.7 How to call an ASM subroutine

This BASIC provides a way of executing external subroutines, via the intrinsic function *USR*.

The programmer needs to store the address of the subroutine to be called in the the work space location reserved for *USR*. In the case of the version for dastaZ80, this is at `0x6148` for the LSB and `0x6149` for the MSB.

This can be done from BASIC with the instruction `DPOKE 24904,<address>`

[14] The command *WIDTH* has been removed because it doesn't have any use in dastaZ80.

Be aware that at location `0x6147` there is stored a *jp* instruction, which is what is executed when the function *USR* is called from BASIC, and therefore it will jump to the subroutine and never come back unless explicitily specified.

If instead your subroutine contains a *ret*urn instruction, or if you are calling dzOS functions, you **MUST** change the *jp* instruction to a *call* instruction.

This can be done from BASIC with the instruction `POKE 24903,205`

Finally, to call the external subroutine, as the *USR* is a function, it returns a parameter and therefore it must be received. Either by assigning the value to a variable (e.g. `A=USR(0)`) or by printing it or by checking it with an *IF*.

- Valid methods how to use *USR*:
  - `A=USR(0)`
  - `IF USR(0)<>0 THEN ...`
  - `PRINT USR(0)`
- Invalid methods:
  - `USR(0)`, will return *?SN Error* (i.e. Syntax Error)

8.7 MS BASIC 4.7b dastaZ80 version

In addition to adapting Grant Searle's version to the dastaZ80 computer, the following has been added:

8.7.1 CAT

Shows a list of BASIC programs in the current disk. Only files of type BASIC are listed, any other files are ignored.

CAT

Parameters: None

Example: `CAT`

8.7.2 COLOUR

Changes the foreground and background colours of the **Low Resolution Display** screen.

COLOUR <*foreground*>,<*background*>

Parameters:

<***foreground***>: number representing one of the available VDP colours.

<***background***>: number representing one of the available VDP colours.

Example: `COLOUR 16,4`

Available VDP colours:

- 0 = Black
- 1 = Red
- 2 = Green
- 3 = Yellow

- 4 = Blue
- 5 = Magenta
- 6 = Cyan
- 7 = White
- 8 = Bright Black (Grey)
- 9 = Bright Red
- 10 = Bright Green
- 11 = Bright Yellow
- 12 = Bright Blue
- 13 = Bright Magenta
- 14 = Bright Cyan
- 15 = Bright White

8.7.3 LOAD

Loads a BASIC program from **DISK** into **MEMORY**.

LOAD "<*filename*>

Parameters:

<***filename***>: the name of the file to be loaded.

Example: `LOAD "mandelbrot`

8.7.4 SAVE

Saves current BASIC program from **MEMORY** into **DISK**.

SAVE "<*filename*>

Parameters:

<***filename***>: the name of the file to be saved.

Example: `SAVE "mandelbrot`

8.7.5 SCREEN

Changes the **Low Resolution Display** screen mode.

SCREEN <*screen_mode*>

Parameters:

<***screen_mode***>: one of the valid **Low Resolution** Screen Modes:

- 0 = Text Mode
- 1 = Graphics I Mode
- 2 = Graphics II Mode
- 3 = Multicolour Mode

- 4 = Graphics II Mode Bitmapped

Example: `SCREEN 2`

8.7.6 SPOKE

Writes a value in a specific **PSG** register.

SPOKE <*PSG_register*>,<*value*>

Parameters:

<***PSG_register***>: **PSG** register number (0-13).

<***value***>: value to set (0..255).

Example: `SPOKE 7,62`

8.7.7 VPEEK

Gets the value at a specific **VRAM** address.

VPEEK <*VRAM_address*>

Parameters:

<***VRAM_address***>: **VRAM** address.

Examples:

- `PRINT VPEEK(6144)`
- `A=VPEEK(6144)`

8.7.8 VPOKE

Writes a value at a specific **VRAM** address.

VPOKE <*VRAM_address*>,<*value*>

Parameters:

<***VRAM_address***>: **VRAM** address.

<***value***>: value to set (0..255).

Example: `VPOKE 6144,171`

8.8 Machine Language Monitor (mlmonitor)

The Machine Language Monitor contains many features that will enable you to create, modfy and test machine language program and subroutines.

It allows to assemble code into memory, disassemble memory, inspect memory, save memory into disk, and more.

Once loaded, the monitor program will display a prompt in the form of a single dot.

At this prompt, the user can enter any of the available commands presented below. Just bear in mind that all commands, addresses and bytes MUST be entered using capital letters. It is highly recommended to activate the *Caps Lock* key during the usage of the monitor program.

Also, all addresses and bytes MUST be entered in hexadecimal notation.

8.8.1 A - Assemble

Purpose: Assemble a line of assembly code.

Syntax: A <*address*> <*mnemonic*> <*operand*>

Example: `A 2000 LD A,(HL)`

<*address*>: a four-digit hexadecimal number indicating the location in memory where to place the generated opcode.

<*mnemonic*>: a valid Z80 assembly language mnemonic (e.g. LD, ADD, CALL, LDIR).

<*operand*>: the operand of the mnemonic.

For a complete list of valid Z80 mnemonics and its operands check the *Z80 Family CPU User Manual*[6], published by ZiLOG.

8.8.2 C - Call

Purpose: Transfers the Program Counter (PC) of the Z80 to the specified address. Hence, the **CPU** starts executing the code it finds at the specified address. Works same as the run command.

Syntax: C <*address*>

Example: `C 4000`

<*address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the byte that will be displayed.

IMPORTANT: In order for the Machine Language Monitor to continue to work after the called subroutine ends, the subroutine MUST end with a return (*RET*) instruction. mlmonitor already takes care of putting in the Stack the current address before changing the Program Counter (PC).

8.8.3 D - Disassemble

Purpose: Disassemble machine code into assembly language mnemonics and operands.

Syntax: D <*start_address*> <*end_address*>

Example: `D 2000 210A`

<*start_address*>: a four-digit hexadecimal number indicating the location in memory of the first byte to disassemble.

<*end_address*>: a four-digit hexadecimal number indicating the location in memory of the last byte to disassemble.

8.8.4 E - Enter program in Hexadecimal

Purpose: Allows to enter Hexadecimal values (e.g. obtained from an assembled program) into memory. It's an *easy* way to test programs.

Syntax: E <*start_address*>

Example: `E 2000`

<*start_address*>: a four-digit hexadecimal number indicating the location in memory of the first address to which hexadecimal values will be inserted.

After entering the command, the ML monitor shows the current value at that address (<start_address>) and allows the user to enter a new value (1 byte). After the user enters a value and presses ENTER, the next address and value will be shown and the process will repeat. To exit (and not change the last value), just press ENTER without entering any value.

8.8.5 F - Fill memory

Purpose: Fill a range of locations with a specified byte.

Syntax: F <*start_address*> <*end_address*> <*byte*>

Example: `F 2000 2100 FF`

<*start_address*>: a four-digit hexadecimal number indicating the location in memory of the first byte to disassemble.

<*end_address*>: a four-digit hexadecimal number indicating the location in memory of the last byte to disassemble.

<*byte*>: the byte value that will be used to fill the locations in memory.

8.8.6 L - Load from disk

Purpose: Load a filename from **DISK** into **RAM** memory. Works similar to the load command, with the difference that here a load address MUST be specified.

Syntax: L <*load_address*> <*filename*>

Example: `L 2000 testfile`

<*load_address*>: a four-digit hexadecimal number indicating the location in memory where the bytes from the filename will start to be stored.

<*filename*>: an existing filename stored in the current **DISK**.

8.8.7 M - Display RAM memory

Purpose: Display **RAM** memory as a hexadecimal dump. Works same as the memdump program.

Syntax: M <*start_address*> <*end_address*>

Example: `M 2000 2100`

<*start_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the first byte to display.

<*end_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the last byte to display.

8.8.8 P - poke

Purpose: Modify a single **RAM** memory address with a specified value. Works same as the poke command.

Syntax: P <*address*> <*byte*>

Example: `P 8000 AB`

<*address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the byte that will be modified.

<*byte*>: the byte value that will be used to modify the location in **RAM** memory.

8.8.9 Q - vpoke

Purpose: Modify a single video **VDP** memory address with a specified value. Works same as the vpoke command.

Syntax: Q <*address*> <*byte*>

Example: `Q 0800 AB`

<*address*>: a four-digit hexadecimal number indicating the location in video **VDP** memory of the byte that will be modified.

<*byte*>: the byte value that will be used to modify the location in video **VDP** memory.

8.8.10 S - Save to disk

Purpose: Save the contents of memory to a filename in **DISK**.

Syntax: S <*start_address*> <*end_address*> <*filename*>

Example: `S 2000 2100 testfile`

<*start_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the first byte that will be saved to **DISK**.

<*end_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the last byte that will be saved to **DISK**.

<*filename*>: a non-existing filename in the current **DISK**.

8.8.11 T - Transfer memory area

Purpose: Transfer segments of **RAM** memory from one memory area to another.

Syntax: T <*start_address*> <*end_address*> <*start_destination_address*>

Example: `T 2000 2100 8000`

<*start_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the first byte that will be transferred.

<*start_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the last byte that will be transferred.

<*start_destination_address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the first byte that will receive the transferred bytes.

8.8.12 V - Display Video RAM memory

Purpose: Display video **VDP** memory as a hexadecimal dump. Works same as the vramdump program.

Syntax: V <*start_address*> <*end_address*>

Example: `V 2000 2100`

<*start_address*>: a four-digit hexadecimal number indicating the location in video **VDP** memory of the first byte to display.

<*end_address*>: a four-digit hexadecimal number indicating the location in video **VDP** memory of the last byte to display.

8.8.13 X - Exit to OS

Purpose: Terminates the Machine Language Monitor and goes back to the Operating System's Command-Line Interface (**CLI**).

Syntax: X

Example: `X`

8.8.14 Y - peek

Purpose: Display the value from a **RAM** memory address. Works same as the peek command.

Syntax: Y <*address*>

Example: `Y 4000`

<*address*>: a four-digit hexadecimal number indicating the location in **RAM** memory of the byte that will be displayed.

8.8.15 Z - vpeek

Purpose: Display the value from a video **VDP** memory address.

Syntax: Z <*address*>

Example: `Z 0800`

<*address*>: a four-digit hexadecimal number indicating the location in video **VDP** memory of the byte that will be displayed.

8.9 Paste File to RAM (pastefile)

Allows to transfer files via a Terminal Emulator software (like Minicom or PuTTY), directly into the **RAM** of the dastaZ80. It's very handy for testing new software under development.

The computer MUST be set to Developer Mode.

The program MUST be run with two parameters:

- RAM address where received bytes will start to be copied.
- Total number of bytes to receive, in hexadecimal 4 digits notation.

For example, lets imagine we have assembled in our Linux PC a program and we are connected to the dastaZ80 with Minicom. The binary of our assembled program has a size of 145 bytes (`0x0091`) and the start address is `0x4420`

First we need to get the hexadecimal values of the assembled binary. There are multiple tools to do this, but from Linux command line we can get it with `xxd -p -c 256`. *-p* outputs the bytes as plain text. *-c 256* prints 256 (the max. we can) bytes per line. Copy the text to the clipboard (typically Ctrl + C). Be sure there are no spaces between the bytes. This will happen if there are more than 256 bytes.

Next, we run pastefile in dastaZ80: `pastefile 4420 0091`

Then we paste the bytes into the terminal program, and pastefile will receive and write them into **RAM**.

It is **important** to remember that you MUST set a character delay of 1ms in your terminal software[15]. Otherwise some bytes will be lost, though dastaZ80 will not be aware. And when you run the program there

[15] In Minicom, press Ctrl + A and then T. The *Terminal settings* will be displayed. Press F, enter a 1 and press ENTER twice.

will be very unexpected results.

9 Appendixes

9.1 Floppy Disk Drive Error Codes

Extracted from: https://github.com/dhansel/ArduinoFDC#troubleshooting

- **0**: No error, the operation succeeded.
- **1**: Internal **ASMDC** error.
- **2**: No disk in drive or drive does not have power.
- **3**: Disk not formatted or not formatted for the correct density.
- **4**: Bad disk or unknown format or misaligned disk drive.
- **5**: Bad disk or unknown format.
- **6**: Bad disk or unknown format.
- **7**: Drive does not have power.
- **8**: Disk is write protected or bad disk.
- **9**: Disk is write protected.

9.2 MS BASIC Error Codes

Error codes are displayed with the message: ?<error_code> Error. For example, ?SN Error for a syntax error.

- **NF**: NEXT without FOR
- **SN**: Syntax error
- **RG**: RETURN without GOSUB
- **OD**: Out of DATA
- **FC**: Function call error
- **OV**: Overflow
- **OM**: Out of memory
- **UL**: Undefined line number
- **BS**: Bad subscript
- **DD**: Re-DIMensioned array
- **DZ**: Division by zero (/0)
- **ID**: Illegal direct
- **TM**: Type miss-match
- **OS**: Out of string space
- **LS**: String too long
- **ST**: String formula too complex
- **CN**: Can't CONTinue
- **UF**: UnDEFined FN function

- **MO**: Missing operand
- **HX**: HEX error
- **BN**: BIN error
- **LE**: File not found while LOADing

9.3 Low Resolution Screen Modes

- Mode 0: **Text Mode**
  - 40 columns by 24 lines.
  - 6x8 bytes characters.
  - 2 colours (Text and Background).
  - No Sprites.
- Mode 1: **Graphics I Mode**
  - 256 by 192 pixels.
  - 32 columns by 24 lines for text.
  - 8x8 bytes characters for text.
  - 15 colours.
  - Sprites.
- Mode 2: **Graphics II Mode**
  - 256 by 192 pixels.
  - 32 columns by 24 lines for text.
  - 8x8 bytes characters for text.
  - 15 colours.
  - Sprites.
- Mode 3: **Multicolour Mode**
  - 64 by 48 pixels.
  - No characters for text.
  - 15 colours.
  - Sprites.
- Mode 4: **Graphics II Mode Bitmapped**: same as mode 2, but screen is bitmapped for addressing every pixel individually.

9.4 Useful pokes

Some of the OS behaviour can be modified by simply changing values stored in **RAM**.

This can be done with the command poke.

- `poke 40AC,01` - Show deleted files with the command cat.

- `poke 40AC,00` - Hide deleted files with the command cat. This is default value when the computer is turned ON or after a reset.
- `poke 4176,nn` - **DISK** number (*nn*) where the operations read and write will be performed.
- `poke 4195,nn` to `poke 41A3,nn` - Change the colour in which messages will be displayed in the **High Resolution Display**. See *dastaZ80 Programmer's Reference Guide*[7] for a complete list of message types.

9.5 How to copy files

At the moment, DZOS doesn't have a command for copying files. But this can be easily achieved using commands load and save.

The *load* command reads bytes from a file in a **DISK** and copies them into **RAM**. On the other hand, the *save* command does exactly the opposite. Hence, we can use these two commands to read the bytes of the file we want to copy, store these bytes in **RAM** and then store these bytes from **RAM** to **DISK** with a new filename.

Let's look at an example: we have a file called *testfile*, which is 956 bytes long, in the disk 1 and we want to copy it to the disk 2.

First, we need to be at the **DISK** unit were the original file is. For our example, it's disk 1. So, if we are not already in disk 1, we use the command `dsk 1`.

Next, we load the file into **RAM** with the command `load testfile`. Once the file is loaded by the OS, it will tell us in which **MEMORY** address was stored. Lets imagine, for this example, that the address was `0x4420`.

Next, we need to position to the destination disk (disk 2 in our example) with the command `dsk 2`.

Finally, we use the command `save 4420 956` to instruct the OS to create a new file with 956 bytes from **RAM**, starting at address **0x4420**. The OS will ask us for a filename, and proceed to perform the saving. For our example we will keep the same name *testfile*.

Now, we have two exact copies of the file *testfile*. One in **DISK** 1 and another in **DISK** 2.

The only downside, a part from the fact that we need to write 3 to 4 commands instead of just one, is that the *File Attributes* will not be the same as the original file, as the OS has just created a new file. We can set the attributes with the command chgattr.

10 Glossary

- **Block Allocation Table (BAT)**: is a data structure used to track disk blocks on a **DISK**. The Table is used as index of files, and each entry in the Table constains details about each file (filename, attributes, size, etc.) and details on how the OS can access the data (e.g. in which Sector the file is stored). Technically, the BAT is not allocating Blocks but Sectors, so it should have been named Sector Allocation Table instead.
- **BIOS**: The Basic Input/Output System is the firmware used to provide hardware related subroutines to the operating system. It is stored in an EEPROM that cannot be modified unless extracting the chip, but during the boot sequence the entire OS (BIOS, Kernel and CLI) are copied into RAM.
- **CLI**: The Command-Line Interface is the part of the operating systems that is in charge of reading and interpreting input entered by the user and outputting information back to the user.
- **Composite video**: Composite video (also knows by CVBS) is an analog video format that combines, on one wire, the video information required to recreate a colour picture, line and frame synchronisation pulses. A yellow RCA connector is typically used for this type of signal.
- **CPU**: A Central Processing Unit is the chip that executes instructions and I/O operations. In the case of the dastaZ80, the CPU is a Zilog Z80 running at 7.3728 Mhz.
- **Disk Image file**: A disk image file contains a snapshot of a bit-by-bit copy of a storage device's structure. In the case of dastaZ80, this structure is defined by the characteristics of the DZFS (dastaZ80 File System). Each image file can be understood as a separate hard disk connected to the computer, but in the form of a file instead of being a physical hard disk drive.
- **DZFS**: dastaZ80 File System is a file system of my own design, for storage devices, aimed at simplicity. It allows Disk Image files of a maximum of 33 MB for a maximum of 1024 files in each image. Each file can be a maximum of 32 KB in size.
- **EEPROM**: Electrically Erasable Programmable Read-Only Memory is a type of non-volatile memory that can be erased and re-programmed.
- **File System**: A file system manages access to the data and the metadata of the files, and the available space of the device, dividing the storage area into units of storage and keeping a map of every storage unit of the device.
- **General-Purpose Input/Output (GPIO)**: is a set of pins which carries **CPU** (among others) signals, and allows external devices to be plugged into the GPIO connector an be managed by the computer. Typical devices connected to a GPIO are; modems, RAM expansions, Real-Time Clocks (RTC), hard drives.
- **Kernel**: The Kernel is a layer of the operating system that it is in between the software and the BIOS. Thus the Kernel is agnostic to the hardware, and is only responsible for calling the needed subroutines.
- **Logical Block Addressing (LBA)**: is a scheme for specifying the location of blocks of data stored on computer storage devices. In DZOS, it is based in blocks of sectors, where each block is 64 sectors a each sector is 512 bytes.
- **LED**: A Light-Emitting Diode is a semiconductor device that emits light when current flows through it. It is commonly used in computers, as light indicator, due to their high durability and low power consumption.
- **NTSC**: National Television System Committee, is the standard for analog television used mainly in USA, Japan and some parts of South America.

- **PAL**: Phase Alternating Line (PAL) is the standard for analog television used mainly in European countries, some African countries, some Asian countries, and some Soth American countries.
- **RAM**: Read-Access Memory is a type of volatile memory, that is generally used for storing temporary data, like for example programs loaded from disk or input entered by the user.
- **RCA connector**: Is a type of electrical connector commonly used to carry audio and video signals. It is also called *phono connector*. The standard is to have yellow connector for video, and white and red for audio.
- **ROM**: Read-Only Memory is a type of non-volatile memory that cannot be electronically modified after the manufacture process.
- **Sector**: A virtual group of 512 bytes stored in a **DISK**.
- **Superblock**: The first 512 bytes on a **DISK** contain fundamental information about the **DISK** geometry, and is used by the OS to know how to access every other information on the **DISK**. On IBM PC-compatibles, this is known as the *Master Boot Record* (MBR). I decided to call it *Superblock*, as it is an orphan Sector that doesn't belong to any Block and it's physically store above any other Block.
- **TTL**: Transistor-Transistor Logic is a family of digital integrated circuits. The important thing to know for the usage of the dastaZ80 is that standard TTL circuits operate with a 5 Volt power supply.
- **VGA**: The Video Graphics Array is a video display controller that became the standard for video output for computers around 1990s, and remained until the introduction of other standards around 2000s.

References

[1] David Asta. *dastaZ80 User's Manual*, 2023.

[2] David Asta. *dastaZ80 Technical Reference Manual*, 2023.

[3] David Asta. dzos github repository. https://github.com/dasta400/dzOS, 2022.

[4] David Asta. Software for dzos github repository. https://github.com/dasta400/dzSoftware, 2022.

[5] The Nascom Microcomputer Division of Lucas Logic Ltd. *Nascom 2 Microcomputer DOCUMENTATION.*

[6] ZiLOG. *Z80 Family CPU User Manual*, um008005-0205 edition, 2004.

[7] David Asta. *dastaZ80 Programmer's Reference Guide*, 2023.

[8] IBM Systems Development Division. *General Information - Binary Synchronous Communications*, first edition.

[9] Carl Lloyd-Parker. Nascom basic disassembled - part 1. *80-Bus News*, 2(3):27–35, 1983.

[10] Carl Lloyd-Parker. Nascom basic disassembled - part 2. *80-Bus News*, 2(4):23–34, 1983.

[11] Carl Lloyd-Parker. Nascom basic disassembled - part 3. *80-Bus News*, 2(5):31–38, 1983.

[12] Carl Lloyd-Parker. Nascom basic disassembled - part 4. *80-Bus News*, 2(6):31–38, 1983.

[13] Carl Lloyd-Parker. Nascom basic disassembled - part 5. *80-Bus News*, 3(1):23–34, 1984.

[14] Carl Lloyd-Parker. Nascom basic disassembled - part 6. *80-Bus News*, 3(2):23–30, 1984.

[15] Carl Lloyd-Parker. Nascom basic disassembled - part 7 - the end. *80-Bus News*, 3(3):23–30, 1984.

[16] Grant Searle. 7-chip z80 computer. http://www.searle.wales/.

[17] Grant Searle. My simple z80 design. http://www.searle.wales/.