dastaZ80 Mark II
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**.

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]

Contents

1 Introduction **1**

2 dastaZ80 Overview **2**

3 Setting up the system **3**
3.1 Transferring software to and from a disk image 4
3.1.1 Copy to a disk image 4
3.1.2 Copy from a disk image 4
3.2 Dual Video Output 4
3.3 Dual Joystick Port 5

4 Operating System (OS) **6**
4.1 dastaZ80 File System (DZFS) 6
4.1.1 DZFS limitations 7
4.2 The Command Prompt 8

5 OS Commands **9**
5.1 General Commands 9
5.1.1 help 9
5.1.2 peek 9
5.1.3 poke 9
5.1.4 autopoke 10
5.1.5 reset 10
5.1.6 halt 10
5.1.7 run 11
5.1.8 crc16 11
5.1.9 clrram 12
5.2 Real-Time Clock (RTC) Commands 12
5.2.1 date 12
5.2.2 time 12
5.2.3 setdate 12
5.2.4 settime 13
5.3 Disk Commands 13
5.3.1 cat 13
5.3.2 erasedsk 14
5.3.3 formatdsk 14
5.3.4 load 15
5.3.5 run 15
5.3.6 rename 15
5.3.7 delete 16
5.3.8 chgattr 16
5.3.9 save 17

5.3.10 dsk . . . 17
5.3.11 diskinfo . . . 17
5.3.12 disklist . . . 18
5.4 VDP (*Low Resolution Screen*) Commands . . . 18
5.4.1 vpoke . . . 18
5.4.2 screen . . . 19

6 Other Software **20**
6.1 Memory Dump (memdump) . . . 20
6.2 Video Memory Dump (vramdump) . . . 20
6.3 Load Screen dumps (loadscr) . . . 21
6.4 Load Font (loadfont) . . . 21
6.5 MS BASIC 4.7b . . . 21
6.5.1 MS BASIC characteristics . . . 21
6.5.2 Speeding up programs . . . 22
6.6 MS BASIC 4.7 Standard version . . . 22
6.6.1 Intrinsic Functions . . . 23
6.6.2 Statements . . . 23
6.6.3 Commands . . . 24
6.6.4 Operators . . . 24
6.6.5 Relational Operators . . . 24
6.6.6 Logical Operators . . . 24
6.6.7 How to call an ASM subroutine . . . 24
6.7 MS BASIC 4.7 dastaZ80 version . . . 25
6.7.1 LOAD . . . 25
6.7.2 SAVE . . . 25
6.7.3 SOUND . . . 26

7 Appendixes **27**
7.1 Floppy Disk Drive Error Codes . . . 27
7.2 MS BASIC Error Codes . . . 27

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, 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 a dastaZ80 computer. Nevertheless, due to its design principles DZOS is portable to other 8-bit machines and hence mostly everything described here is applicable to DZOS running on other computers.

2 dastaZ80 Overview

- Zilog Z80 microprocessor (CPU) running at 7.3728 MHz
- 64 KB **MEMORY**
  - 16 KB reserved for DZOS
  - 48 KB available for the user
  - 56 Bytes NVRAM
- Storage devices
  - Micro SD Card, formatted with FAT32 and containing Disk Image Files formatted with DZFS[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
  - Programmable Sound Generator (PSG) with 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, provides valid until the year 2100 leap year compensation.
- 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
  - 2x20 pin male header. Exposes all CPU signals

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

[2] Same chip as used in numerous Arcade machines and in Amstrad CPC, Atari ST, MSX, ZX Spectrum and other home computers.

3 Setting up the system

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 Battery Button Cell
- Optionally, a monitor with NTSC Composite input, and a 3.5mm jack to 3 RCA Audio/Video cable[3].

Steps:

1. Insert the battery LIR2032 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:
   - 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.
6. 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.
7. 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.
8. Connect the female power supply connector to the male connector at the back of the case.

That's really it. Switch the computer on, with the power switch, and you should see text on your monitor. DZOS is ready to use.

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

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.

3.1 Transferring software to and from a disk image

3.1.1 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[4]. 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*.

3.1.2 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.

3.2 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.

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

The Composite output, called *Low Resolution*, can only provide 40 columns in Text Mode and 32 in Graphics Modes[5], 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.

3.3 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.

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

4 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.

4.1 dastaZ80 File System (DZFS)

A file system manages access to the data stored in a storage medium, a SD card in the case of dastaZ80, and allows the OS to load and save data in the SD card. 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 system based in blocks of sectors. The allocation table is called Block Allocation Table (BAT).

Without entering into much details, bytes in the **DISK** are organised as Sectors, and Sectors are grouped into Blocks. Each Sector is 512 bytes long and each Block holds 64 Sectors. Therefore, a Block is 32,768 bytes long (64 * 512).

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.

4.1.1 DZFS limitations

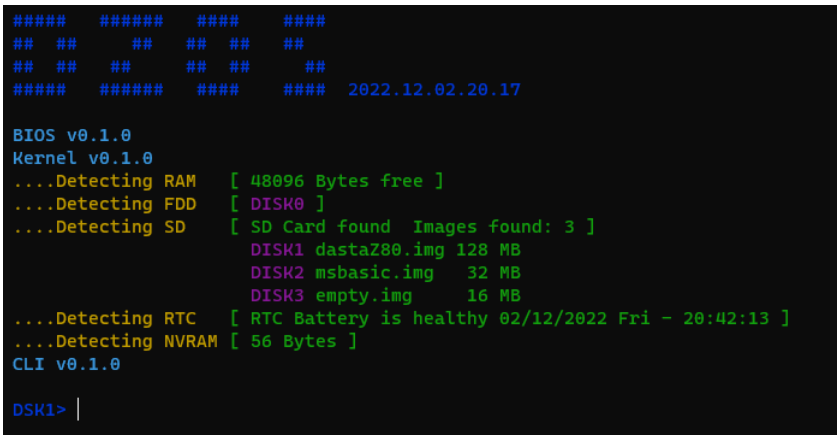
The current version of the DZFS implementation (DZFSV1) have the following limitations:

- 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 there is a separate field for File Type.
- Maximum size for a file is 32,768 bytes.

4.2 The Command Prompt

When you switch ON the computer, you will hear a low tone (beep) and the **Low Resolution Display** will show the computer logo:

the **High Resolution Display**, will show some information on the screen:



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**.

5 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.

5.1 General Commands

5.1.1 help

Shows a list of the most important commands available for the user. For a complete list of commands, refer always to this manual.

> **help**

Parameters: None

5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 reset

Performs a reset. It has the same effect as pressing the **RESET** button at the side of the computer.

> **reset**

Parameters: None

5.1.6 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.

IMPORTANT: **to use the computer again, you MUST turn it off and on again. Do NOT just press the reset button**. Otherwise corruption of data will occur, because the **DISK** controller is only reset at power on, and not when the **RESET** button is pressed.

> **halt**

Parameters: None

5.1.7 run

There are two formats of this command:

run <***address***> and **run** <***filename***>

Here the former is described. See the section *5.3 DISK Commands* on page 13 for the other format.

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

Parameters:

address: address from where to start running.

Example: > `run 4420`

Will transfer the Program Counter (PC) of the Z80 to the address 0x4420. Therefore, 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*[6]. Otherwise the user will have to reset the computer to get back to CLI. Not harmful but cumbersome.

5.1.8 crc16

Generates a CRC-16/BUYPASS1[6]

There are two formats of this command:

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

Here the former is described. See the section *5.3 DISK Commands* on page 13 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
```

[6] A 16-bit cyclic redundancy check (CRC) based on the IBM Binary Synchronous Communications protocol[7] (BSC or Bisync). It uses the polynomial $X^{16} + X^{15} + X^2 + 1$

5.1.9 clrram

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

> **clrram**

Parameters: None

5.2 Real-Time Clock (RTC) Commands

5.2.1 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
```

5.2.2 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
```

5.2.3 setdate

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

> **setdate** <***yyyy***><***mm***><***dd***><***dow***>

Parameters:

yyyy: year (2 digits).

mm: month.

dd: day.

yyyy: dow (Day of the Week. 1=Sunday).

Example: > `setdate 2211032`

5.2.4 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`

5.3 Disk Commands

5.3.1 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
```

The File Types are:

- **USR**: User defined.
- **EXE**: Executable binary.
- **BIN**: Binary (non-executable) data.
- **BAS**: BASIC code.

- **TXT**: Plain ASCII Text file.
- **FN6**: Font (6×8) for Text Mode.
- **FN8**: Font (8×8) for Graphics Modes.
- **SC1**: Screen 1 (Graphics I Mode) Picture.
- **SC2**: Screen 2 (Graphics II Mode) Picture.
- **SC3**: Screen 3 (Multicolour Mode) Picture.

5.3.2 erasedsk

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

This is a destructive action 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`

5.3.3 formatdsk

Formats a **DISK** with DZFS format. This is a destructive action 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.

5.3.4 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.

5.3.5 run

There are two formats of this command:

run <***address***> and **run** <***filename***>

Here the latter is described. See the section *5.1 General Commands* on page 9 for the other format.

> **run** <***filename***>

Parameters:

filename: the name of the file to run.

Example: > `run HelloWorld`

Will load the contents of the file *HelloWorld* into **MEMORY**, and then transfer the Program Counter (PC) of the Z80 to the address stored in the BAT entry. Therefore, the CPU will start running whatever instructions finds from the address 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*[6]. Otherwise the user will have to reset the computer to get back to CLI.

5.3.6 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*.

5.3.7 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 [7] 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.

Example: > `delete HelloWorld`

Will delete the file *HelloWorld*.

5.3.8 chgattr

Files in DZFS 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* <*filename*>.

> **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

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

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.

5.3.9 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.

5.3.10 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.

5.3.11 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
```

5.3.12 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 the unlikely case that you may have a program loaded that uses those low bytes, after executing the *disklist* command the program will be corrupted.

5.4 VDP (*Low Resolution Screen*) Commands

5.4.1 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.

5.4.2 screen

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

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

Parameters:

mode: one of the following values:

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

Example: `> screen 0`

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

6 Other Software

6.1 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
```

6.2 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
```

6.3 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).

6.4 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.

6.5 MS BASIC 4.7b

The Nascom 2 computer came with MS BASIC 7.4 installed in ROM, and the disassembled code was published in the *80-BUS NEWS* magazine [8], [9], [10], [11], [12], [13], [14].

Grant Searle published a modification in his Grant's *7-chip Z80 computer* webpage[15].

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

6.5.1 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

6.5.2 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.
- 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.

6.6 MS BASIC 4.7 Standard version

Standard version is referring to the version adapted by Grant Searle [16] of the original NASCOM 2 version.

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

6.6.1 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$**: Returns a string whose one element has its ASCII code
- **COS**: Returns cosine in radians
- **EXP**: Returns a number to the power of another number
- **FRE**: Returns number of bytes in memory not being used by BASIC.
- **INP**: Reads a byte from a port
- **INT**: Returns the largest integer of a floating number
- **LEFT$**: 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

6.6.2 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**

6.6.3 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

6.6.4 Operators

- +, -, *, /, ˆ

6.6.5 Relational Operators

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

6.6.6 Logical Operators

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

6.6.7 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>`

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)

6.7 MS BASIC 4.7 dastaZ80 version

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

6.7.1 LOAD

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

LOAD "<*filename*>

Parameters:

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

Example: `LOAD "mandelbrot`

6.7.2 SAVE

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

SAVE "<*filename*>

Parameters:

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

Example: `SAVE "mandelbrot`

6.7.3 SOUND

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

SOUND <***PSG_register***>,<***value***>

Parameters:

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

<***value***>: value to set (0.255).

Example: `SOUND 7,62`

7 Appendixes

7.1 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.

7.2 MS BASIC Error Codes

- **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

References

[1] David Asta. *dastaZ80 User's Manual*, 2022.

[2] David Asta. *dastaZ80 Technical Reference Manual*, 2022.

[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] David Asta. *dastaZ80 Programmer's Reference Guide*, 2022.

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

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

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

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

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

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

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

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

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

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