

# **dastaZ80**

## **Mark I**

**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

All **hardware** is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License** (<http://creativecommons.org/licenses/by-sa/4.0/>)

All **software** is licensed under **The MIT License** (<https://opensource.org/licenses/MIT>)

All **documentation** is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License** (<http://creativecommons.org/licenses/by-sa/4.0/>).

---

Copyright 2012-2022 David Asta

## Table of Contents

1	Introduction.....	3
2	Document Conventions.....	4
3	Related Documentation.....	5
4	dastaZ80 Overview.....	6
5	Setting up the system.....	7
6	Operating System (OS).....	8
6.1	dastaZ80 File System (DZFS).....	8
6.1.1	DZFS limitations.....	9
6.2	The Command Prompt.....	10
6.3	OS Commands.....	11
6.3.1	General Commands.....	11
6.3.1.1	help.....	11
6.3.1.2	peek.....	11
6.3.1.3	poke.....	12
6.3.1.4	autopoke.....	12
6.3.1.5	memdump.....	13
6.3.1.6	reset.....	13
6.3.1.7	halt.....	14
6.3.1.8	run.....	14
6.3.1.9	crc16.....	15
6.3.2	Real-Time Clock Commands.....	16
6.3.2.1	date.....	16
6.3.2.2	time.....	16
6.3.2.3	setdate.....	16
6.3.2.4	settime.....	17
6.3.3	DISK Commands.....	18
6.3.3.1	cat.....	18
6.3.3.2	formatdisk.....	18
6.3.3.3	load.....	19
6.3.3.4	run.....	19
6.3.3.5	rename.....	20
6.3.3.6	delete.....	21
6.3.3.7	chgattr.....	22
6.3.3.8	save.....	23
6.3.3.9	diskinfo.....	23
6.3.3.10	crc16.....	24

# 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 Document Conventions

The following conventions are used in this manual:

<b>MUST</b>	MUST denotes that the definition is and absolute requirement.
<b>SHOULD</b>	SHOULD denotes that it is recommended, but that there may exist valid reasons to ignore it.
<b>DEVICE</b>	Device names are displayed in bold all upper case letters, and refer to hardware devices.
<b>command</b>	Operating System command keywords are displayed in bold all lower case letters.
<i>&lt;text&gt;</i>	Angle brackets enclose variable information that you <b>MUST</b> supply. In place of <i>&lt;text&gt;</i> , substitute the value desired. Do not enter the angle brackets when entering the value.
[ <i>text</i> ]	Square brackets enclose variable information that you <b>COULD</b> supply. They are optional. In place of [ <i>text</i> ], substitute the value desired. Do not enter the square brackets when entering the value.
FreeMono	Text appearing in the FreeMono font represents information that you type in via the keyboard.
0x0000	Numbers prefixed by 0x indicate an Hexadecimal value.
0x14B0	Unless specified, memory addresses are always expressed in Hexadecimal.
0x00FE	
Return	Refers to the key Return in the keyboard.

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

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

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

### **3 Related Documentation**

*dastaZ80 Programmer's Reference Guide*

*dastaZ80 Technical Reference Manual*

*dastaZ80 Appendixes*

<https://github.com/dasta400/dzOS>

## 4 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
- Storage device
  - CompactFlash Card, formatted with DZFS<sup>1</sup>
- Video output
  - VGA 80 column by 25 lines and 16 colours
- Keyboard
  - Acorn Archimedes A3010 keyboard. 102 keys with 12 function keys, cursor keys and keypad
- Case
  - Repurposed Acorn Archimedes A3010 case with keyboard.
- Expansion ports
  - 2x20 pin male header. Exposes all CPU signals.

---

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

## 5 Setting up the system

Setting up the dastaZ80 is very easy.

You will only need:

- the dastaZ80 computer.
  - A 9 to 12 volts power supply with a female 2.1mm barrel-style DC connector (positive polarity).
  - A CompactFlash card.
  - A VGA monitor.
1. Introduce the CompactFlash card in the CF slot at the back of the computer case. This procedure **MUST** be performed with the computer switched off, and **SHOULD** be performed with the computer unplugged.
  2. 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.
  3. 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.



## 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 CompactFlash (CF) card in the case of dastaZ80, and allows the OS to load and save data in the CF 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.

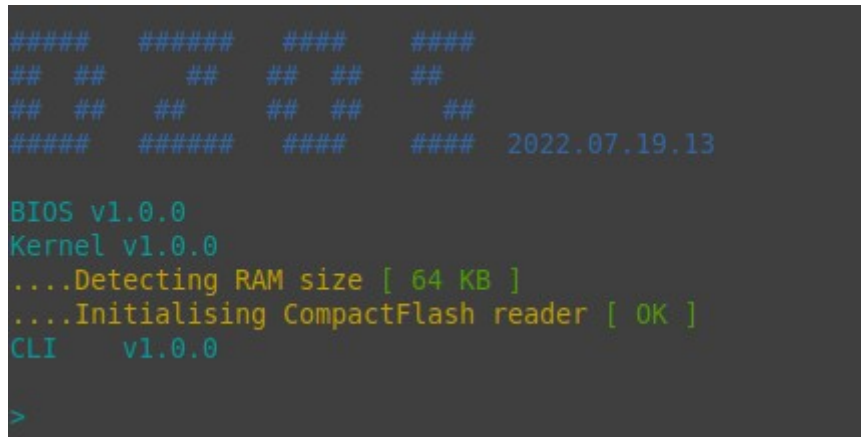
### 6.1.1 DZFS limitations

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.
- Maximum size for a file is 32,768 bytes.

## 6.2 The Command Prompt

When you switch on the computer, you will see some information on the screen.



```
#####  #####  ####  ####  
##  ##      ##  ##  ##  
##  ##      ##  ##  ##  
#####  #####  ####  ####  2022.07.19.13  
  
BIOS v1.0.0  
Kernel v1.0.0  
....Detecting RAM size [ 64 KB ]  
....Initialising CompactFlash reader [ OK ]  
CLI v1.0.0  
  
>
```

This information tells you about the release version of DZOS (2022.07.19.13 in the screenshot). The BIOS, Kernel and CLI versions and some processes like detecting **RAM** and initialising the **DISK**.

After that information, you'll see the symbol >

This symbol is called the *command prompt*. To the right of the command prompt you will see a flashing underscore. This is called the *cursor*. The cursor shows where the command you type will appear.

## 6.3 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.

### 6.3.1 General Commands

#### 6.3.1.1 **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

#### 6.3.1.2 **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.

### 6.3.1.3 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.

### 6.3.1.4 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*: first 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.

### 6.3.1.5 memdump

Shows the **MEMORY** 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).

```
> memdump <start_address>, <end_address>
```

#### Parameters:

*start\_address*: first address where the user wants to get values from.

*end\_address*: last address where the user wants to get values from.

**Example:** > memdump 0B40,0BEF

Will show:

```

      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  .....
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
```

### 6.3.1.6 reset

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

```
> reset
```

**Parameters:** None

### 6.3.1.7 halt

Disables interrupts and puts the CPU in halted state, effectively making the computer unusable until next reboot. SHOULD be used before switching the computer off, to ensure no changes are happening. MUST not be used while the busy light of the **DISK** is on.

```
> halt
```

**Parameters:** None

### 6.3.1.8 run

There are two formats of this command: **run** <address> and **run** <filename>. Here the former is described. See the section 6.3.3 DISK Commands for the other format.

Starts running instructions from a specific **MEMORY** address.

```
> run <address>
```

**Parameters:**

*address*: from where to start running.

**Example:** > run 2100

Will transfer the Program Counter (PC) of the Z80 to the address 0x2100. Therefore, the CPU will start running whatever instructions finds from 0x2100 and on. Programs run this way MUST end with a jump instruction (JP) to CLI prompt address, as described in the *dastaZ80 Programmer's Reference Guide*. Otherwise the user will have to reset the computer to get back to CLI. Not harmful but cumbersome.

### 6.3.1.9 **crc16**

Generates a CRC-16/BUYPASS<sup>2</sup>

There are two formats of this command: **crc16** <start\_address>, <end\_address> and **crc16** <filename> Here the former is described. See the section 6.3.3 DISK Commands 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.

*start\_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
```

---

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



## 6.3.2 Real-Time Clock Commands

### 6.3.2.1 `date`

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

```
> date
```

**Parameters:** None

**Example:** `> date`

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

```
Today: 12/08/2022
```

### 6.3.2.2 `time`

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

```
> time
```

**Parameters:** None

**Example:** `> time`

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

```
Now: 16:24:45
```

### 6.3.2.3 `setdate`

Changes the date stored in the **RTC**

```
> setdate <new_date>
```

**Parameters:**

*new\_date*: new date to be stored. The format is DDMMYYYY.

**Example:** `> setdate 31122022`

## 6.3.2.4 **settime**

Changes the time stored in the **RTC**

```
> settime <new_time>
```

### **Parameters:**

*new\_time*: new time to be stored. The format is HHMMSS, and HH is 24h format.

**Example:** > **settime** 145600

## 6.3.3 DISK Commands

### 6.3.3.1 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	Created		Modified		Size	Attributes
HelloWorld	12-03-2022	13:21:44	12-03-2022	13:21:22	000038	E

### 6.3.3.2 formatdsk

Formats a **DISK** with DZFS format. This is a destructive action and all data in the **DISK** will be destroyed and make it unreadable by other computers that do not support DZFS.

```
> formatdsk <label>, <partitions>
```

**Parameters:**

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

*partitions*: currently DZFSV1 only supports one partition, so this number MUST be a 1.

**Example:**

```
> formatdsk mainDisk,1
```

Will format the CompactFlash card inserted in the CF slot at the back of the computer case with one partition, having *mainDisk* as disk label.

### 6.3.3.3 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 myfile1
```

Will load the contents (bytes) of the file *myfile1* and copy them into the **RAM** address from which it was originally saved.

### 6.3.3.4 run

There are two formats of this command: **run** <address> and **run** <filename> Here the latter is described. See the section 6.3.1 General Commands for the other format.

Loads and runs a file stored in the **DISK**.

```
> run <filename>
```

#### Parameters:

*filename*: the name of the file to run.

**Example:** > run msbasic

Will load the contents of the file *msbasic* 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 on. Programs run this way **MUST** end with a jump instruction (JP) to CLI prompt address, as described in the *dastaZ80 Programmer's Reference Guide*. Otherwise the user will have to reset the computer to get back to CLI.

### 6.3.3.5 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 myfile1, readme
```

Will change the name of the file *myfile1* to *readme*.

### 6.3.3.6 delete

Deletes a file from the **DISK**.

Technically is not deleting anything but just changing the first character of the filename to ~, 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<sup>3</sup> 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 readme
```

Will delete the file *readme*.

---

<sup>3</sup> By free space on the **DISK** we understand a Block in the DZFS BAT that was never used before by a file.

### 6.3.3.7 **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 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 readme, RE
```

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

### 6.3.3.8 **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**.

The user will be prompted to enter a filename, which MUST follow the limitations explained in the section 6.1.1 DZFS limitations of this manual.

**Example:**

```
> save 4570, 512
```

Will create a new file, with the name entered by the user when prompted, with 512 bytes of the contents of **MEMORY** from 0x4570 to 0x476F.

### 6.3.3.9 **diskinfo**

Shows some information about the **DISK**.

```
> diskinfo
```

**Parameters:** None

**Example:**

```
> diskinfo
```

Will show (information will differ depending on your **DISK**):

```
CompactFlash 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
```



### 6.3.3.10 **crc16**

Generates a CRC-16/BUYPASS<sup>4</sup>

There are two formats of this command: **crc16** <start\_address>, <end\_address> and **crc16** <filename> Here the latter is described. See the section 6.3.1 General Commands for the other format.

```
> crc16 <filename>
```

**Parameters:**

*filename*: the name of the file for which the CRC will be calculated.

**Example:** > **crc16** HelloWorld

Will calculate the CRC of all bytes on the file and show it on the screen:

```
CRC16: 0x3ABC
```

---

<sup>4</sup> A 16-bit cyclic redundancy check (CRC) based on the IBM Binary Synchronous Communications protocol (BSC or Bisync). It uses the polynomial  $X^{16} + X^{15} + X^2 + 1$