dastaZ80 Mark I
Programmer's Reference Guide

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.
`Courier`	Text appearing in the `Courier` font represents either an OS System Variable[1], a Z80 CPU Register or a Z80 Flag.
`0x14B0`	Numbers prefixed by 0x indicate an Hexadecimal value. Unless specified, memory addresses are always expressed in Hexadecimal.
F_abcdef	Text starting with F_ refers to the name of an OS routine that can be called via Jumpblocks.
jp abcdef	Refers to the Z80 mnemonic for *jump*, which transfers the CPU Program Counter to a specific **MEMORY** address.

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

The 80 column VGA output is referred as **CONSOLE**.

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

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

In the list of routines, the **Destroys** lists the **CPU** registers and **MEMORY** System Variables that are destroyed by the routine in question. But bare in mind that a routine may call other routines that may destroy other registers and variables. Refer to the **Calls** list to check the entire flow. By *Destroys* is understood that the listed register or variable value is overwritten within the routine.

FIXME ???? WHERE IS THE FOOTNOTE ????

Related Documentation

dastaZ80 User's Manual
dastaZ80 Technical Reference Manual
https://github.com/dasta400/dzOS

Contents

1 Memory Map

1.1 ROM

Address		Description		Size (bytes)
0x0008	0x01D9	init SIO/2	**BIOS**	466
0x01DA	0x133F	BIOS code		4,462
0x1340	0x13BF	BIOS Jumpblock		128
0x13C0	0x267F	Kernel code	**Kernel**	4,800
0x2670	0x267F	dzOS version build		16
0x2680	0x277F	Kernel Jumpblock		256
0x2780	0x3B3F	CLI code	**CLI**	5,056
0x3B40	0x3C3F	Bootstrap	**BOOTSTRAP**	256
0x3C40	0x3FFF	Free		960

1.2 RAM

Address		Description		Size (bytes)
0x4000	0x401F	**Stack**		32
0x4020	0x4174	**System Variables**		341
SIO		0x4020	SIO_CH_A_BUFFER	64
		0x4060	SIO_CH_A_IN_PTR	2
		0x4062	SIO_CH_A_RD_PTR	2
		0x4064	SIO_CH_A_BUFFER_USED	1
		0x4065	SIO_CH_B_BUFFER	64
		0x40A5	SIO_CH_B_IN_PTR	2
		0x40A7	SIO_CH_B_RD_PTR	2
		0x40A9	SIO_CH_B_BUFFER_USED	1
		0x40AA	SIO_PRIMARY_IO	1
CF Superblock		0x40AB	CF_is_formatted	1
		0x40AC	CF_cur_partition	1
		0x40AD	CF_cur_sector	2
CF BAT		0x40BD	CF_cur_file_attribs	1
		0x40BE	CF_cur_file_time_created	2
		0x40C0	CF_cur_file_date_created	2
		0x40C2	CF_cur_file_time_modified	2
		0x40C4	CF_cur_file_date_modified	2
		0x40C6	CF_cur_file_size_bytes	2
		0x40C8	CF_cur_file_size_sectors	1
		0x40C9	CF_cur_file_entry_number	2
		0x40CB	CF_cur_file_1st_sector	2
		0x40CD	CF_cur_file_load_addr	2
CLI		0x40CF	CLI_buffer_cmd	16
		0x40DF	CLI_buffer_parm1_val	16

Address		Description		Size (bytes)
		0x40EF	CLI_buffer_parm2_val	16
		0x40FF	CLI_buffer_pgm	32
		0x411F	CLI_buffer_full_cmd	64
Generic		0x415F	tmp_addr1	2
		0x4161	tmp_addr2	2
		0x4163	tmp_addr3	2
		0x4165	tmp_byte	1
RTC		0x4166	RTC_hour	1
		0x4167	RTC_minutes	1
		0x4168	RTC_seconds	1
		0x4169	RTC_century	1
		0x416A	RTC_year	1
		0x416B	RTC_year4	2
		0x416D	RTC_month	1
		0x416E	RTC_day	1
		0x416F	RTC_day_of_the_week	1
NVRAM (RTC)		0x4170	NVRAM_battery_status	1
Math		0x4171	MATH_CRC	2
		0x4173	MATH_polynomial	2
0x4175	0x421F	**Reserved for future use**		171
0x4220	0x441F	**DISK Buffer**		512
0x4420	0xFFFF	**Free RAM**		48,096

2 I/O Map

ROM Paging	`0x38`	
SIO	`0x80`	Channel A Control
	`0x81`	Channel A Data
	`0x82`	Channel B Control
	`0x83`	Channel B Data
CompactFlash Card	`0x10`	

3 BIOS Jumpblocks

3.1 General Routines

3.1.1 F_BIOS_CBOOT

Action	Cold Boot. Executed when the computer is powered on, or after a reset by pressing the **RESET** push-button
Entry	None
Exit	None
Destroys	None
Calls	*jp* F_BOOSTRAP_START

3.1.2 F_BIOS_WBOOT

Action	Warm Boot. Executed after **SIO/2** initialisation, or after a *reset* command
Entry	None
Exit	None
Destroys	None
Calls	*jp* F_KRN_START

3.1.3 F_BIOS_SYSHALT

Action	Halts the computer. Executed after a *halt* command
Entry	None
Exit	Disables Interrupts (di)
Destroys	None
Calls	None

3.2 Serial Routines

3.2.1 F_BIOS_SERIAL_INIT

Action	Initialises **SIO/2**: sets Channels A and B as 115,000 bps, 8N1, Interrupt in all characters Configures the interrupt vector to 0x60 Sets the CPU to Interrupt Mode 2 Enables Interrupts
Entry	None
Exit	None
Destroys	`A`, `HL`
Calls	*jp* F_BIOS_WBOOT

3.2.2 F_BIOS_SERIAL_CONIN_A

Action	Reads a character from the **SIO/2** Channel A
Entry	None
Exit	`A` = character read
Destroys	`A`
Calls	None

3.2.3 F_BIOS_SERIAL_CONIN_B

Action	Reads a character from the **SIO/2** Channel B
Entry	None
Exit	`A` = character read
Destroys	`A`
Calls	None

3.2.4 F_BIOS_SERIAL_CONOUT_A

Action	Sends a character to the **SIO/2** Channel A
Entry	`A` = character to be send
Exit	None
Destroys	None
Calls	None

3.2.5 F_BIOS_SERIAL_CONOUT_B

Action	Sends a character to the **SIO/2** Channel B
Entry	`A` = character to be send
Exit	None
Destroys	None
Calls	None

3.3 Compact-Flash (DISK) Routines

3.3.1 F_BIOS_CF_INIT

Action	Sets **DISK** to 8-bit data transfer mode
Entry	None
Exit	None
Destroys	`A`
Calls	F_BIOS_CF_BUSY

3.3.2 F_BIOS_CF_BUSY

Action	Checks if the **DISK** busy bit (0=ready, 1=busy) and loops until it is not busy
Entry	None
Exit	None
Destroys	`A`
Calls	None

3.3.3 F_BIOS_CF_DISKINFO

Action	Executes an *Identify Drive* command
Entry	None
Exit	None
Destroys	`A`, `B`, `HL`, `CF_BUFFER_START`
Calls	F_BIOS_CF_BUSY

3.3.4 F_BIOS_CF_SET_LBA

Action	Sets Sector count and LBA address
Entry	`E` = sector address LBA 0 (bits 0-7) `D` = sector address LBA 1 (bits 8-15) `C` = sector address LBA 2 (bits 16-23) `B` = sector address LBA 3 (bits 24-27)
Exit	None
Destroys	`A`
Calls	F_BIOS_CF_BUSY

3.3.5 F_BIOS_CF_READ_SEC

Action	Reads a Sector (512 bytes), from the **DISK** and places the bytes into the `CF_BUFFER_START`
Entry	`E` = sector address LBA 0 (bits 0-7) `D` = sector address LBA 1 (bits 8-15) `C` = sector address LBA 2 (bits 16-23) `B` = sector address LBA 3 (bits 24-27)
Exit	`CF_BUFFER_START` contains the 512 bytes read
Destroys	`A`, `B`, `HL`, `CF_BUFFER_START`
Calls	F_BIOS_CF_SET_LBA F_BIOS_CF_BUSY

3.3.6 F_BIOS_CF_WRITE_SEC

Action	Writes a Sector (512 bytes), from the `CF_BUFFER_START` into the **DISK**
Entry	`E` = sector address LBA 0 (bits 0-7) `D` = sector address LBA 1 (bits 8-15) `C` = sector address LBA 2 (bits 16-23) `B` = sector address LBA 3 (bits 24-27)
Exit	`CF_BUFFER_START` contains the 512 bytes written
Destroys	`A`, `B`, `HL`
Calls	F_BIOS_CF_SET_LBA F_BIOS_CF_BUSY

4 Kernel Jumpblocks

4.1 Serial Routines

4.1.1 F_KRN_SERIAL_WRSTR

Action	Outputs a string, terminated with Carriage Return to the **CONSOLE**.
Entry	`HL` = address in **MEMORY** where the first character of the string to be output is.
Exit	None
Destroys	`A`, `HL`
Calls	F_BIOS_SERIAL_CONOUT_A

4.1.2 F_KRN_SERIAL_SETFGCOLR

Action	Set the colour that will be used for the foreground (text). The colour will remain until a different one is set.
Entry	`A` = Colour number (as listed in Appendixes section)
Exit	None
Destroys	`B`, `DE`
Calls	F_BIOS_SERIAL_CONOUT_A *jp* F_KRN_SERIAL_SEND_ANSI_CODE

4.1.3 F_KRN_SERIAL_WRSTRCLR

Action	Outputs a string, terminated with Carriage Return to the **CONSOLE**, with a specific foreground colour.
Entry	`A` = Colour number (as listed in Appendixes section) `HL` = address in **MEMORY** where the first character of the string to be output is.
Exit	None
Destroys	`B`, `DE`
Calls	F_KRN_SERIAL_SETFGCOLR *jp* F_KRN_SERIAL_WRSTR

4.1.4 F_KRN_SERIAL_WR6DIG_NOLZEROS

Action	Outputs to the **CONSOLE** a string of ASCII characters representing a number, without outputing the leading zeros. (.e.g. 30 30 31 32 30 34 is 001204, but the output wil be 1024)
Entry	`IX` = address in **MEMORY** where the ASCII characters are stored.
Exit	None
Destroys	`A`, `B`, `DE`, `IX`
Calls	F_BIOS_SERIAL_CONOUT_A

4.1.5 F_KRN_SERIAL_RDCHARECHO

Action	Reads with echo. Reads a character from the **SIO/2** Channel A, and outputs it to the **CONSOLE**.
Entry	None
Exit	`A` = read character.
Destroys	None
Calls	F_BIOS_SERIAL_CONIN_A F_BIOS_SERIAL_CONOUT_A

4.1.6 F_KRN_SERIAL_EMPTYLINES

Action	Outputs n number of empty lines to the **CONSOLE**.
Entry	`B` = number (n) of empty lines to output.
Exit	None
Destroys	`A`
Calls	F_BIOS_SERIAL_CONOUT_A

4.1.7 F_KRN_SERIAL_PRN_NIBBLE

Action	Outputs a single hexadecimal nibble in hexadecimal notation.
Entry	`A` = nibble to output. Nibble will be the less significant 4 bits of the byte.
Exit	None
Destroys	`A`
Calls	F_BIOS_SERIAL_CONOUT_A

4.1.8 F_KRN_SERIAL_PRN_BYTE

Action	Outputs a single hexadecimal byte in hexadecimal notation.
Entry	`A` = byte to output.
Exit	None
Destroys	`A`
Calls	F_BIOS_SERIAL_CONOUT_A

4.1.9 F_KRN_SERIAL_PRN_BYTES

Action	Outputs n number of bytes as ASCII characters.
Entry	`B` = number (n) of bytes to output. `HL` = address in **MEMORY** where the first byte to output is.
Exit	None
Destroys	`A`, `HL`
Calls	F_BIOS_SERIAL_CONOUT_A

4.1.10 F_KRN_SERIAL_PRN_WORD

Action	Outputs the 4 hexadecimal digits of a word in hexadecimal notation.
Entry	`HL` = word to be output.
Exit	None
Destroys	`A`
Calls	F_KRN_SERIAL_PRN_BYTE

4.1.11 F_KRN_SERIAL_BACKSPACE

Action	
Entry	
Exit	
Destroys	
Calls	

4.1.12 F_KRN_SERIAL_SEND_ANSI_CODE

Action	Writes an ANSI code to the **SIO/2** Channel A.
Entry	`DE` = address in **MEMORY** where the first byte of ANSI escape code is. `B` = number of bytes in the ANSI escape code.
Exit	None
Destroys	`A`, `DE`
Calls	F_BIOS_SERIAL_CONOUT_A

4.2 DZFS (file system) Routines

4.2.1 F_KRN_DZFS_READ_SUPERBLOCK

Action	Reads 512 bytes from Sector 0 (corresponding to the DZFS *Superblock*) into the CF buffer in **MEMORY**. If the *Superblock* does not containe the correct DZFS signature, `CF_is_formatted` is set to 0x00. Otherwise, is set to 0x01.
Entry	None
Exit	None
Destroys	`A`, `BC`, `DE`
Calls	F_BIOS_CF_READ_SEC

4.2.2 F_KRN_DZFS_READ_BAT_SECTOR

Action	Reads a BAT Sector into `CF Card Buffer` in **MEMORY**.
Entry	`CF_cur_sector` holds the sector number for the BAT.
Exit	`CF Card Buffer` contains the BAT sector.
Destroys	`BC`, `HL`
Calls	F_BIOS_CF_READ_SEC

4.2.3 F_KRN_DZFS_BATENTRY_TO_BUFFER

Action	Extracts the data of a BAT entry from the `CF Card Buffer` in **MEMORY** and populates the values into System variables.
Entry	`A` = BAT entry to extract data from.
Exit	CF BAT System Variables are populated. See RAM Memory Map for for details.
Destroys	`A`, `BC`, `DE`, `HL`, `IX`, `tmp_addr1`
Calls	F_KRN_MULTIPLY816_SLOW

4.2.4 F_KRN_DZFS_SEC_TO_BUFFER

Action	Loads a Sector (512 bytes) from the **DISK** and copies the bytes into the `CF Card Buffer` in **MEMORY**.
Entry	`HL` = Sector number to load.
Exit	`CF Card Buffer` contains the bytes of Sector loaded.
Destroys	`DE`, `HL`
Calls	F_BIOS_CF_READ_SEC

4.2.5 F_KRN_DZFS_GET_FILE_BATENTRY

Action	Gets the BAT's entry number of a specified filename.
Entry	`HL` = Address where the filename to check is stored
Exit	BAT Entry values are stored in the SYSVARS. `DE` = $0000 if filename found. Otherwise, whatever value had at start.
Destroys	`A`, `DE`, `HL`, `tmp_addr2`, `tmp_addr3`
Calls	F_KRN_DZFS_READ_BAT_SECTOR F_KRN_DZFS_BATENTRY_TO_BUFFER F_KRN_STRLEN F_KRN_STRCMP

4.2.6 F_KRN_DZFS_LOAD_FILE_TO_RAM

Action	Load a file from **DISK**. Copies the bytes stored in the **DISK** into **MEMORY**, at the specified **MEMORY** address in the BAT.
Entry	`DE` = 1st sector number in the **DISK**. `IX` = file length in sectors.
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`, `IX`, `tmp_addr1`
Calls	F_BIOS_CF_READ_SEC

4.2.7 F_KRN_DZFS_DELETE_FILE

Action	Marks a file as deleted. The mark is done by changing the first character of the filename to 0x7E ( )
Entry	`DE` = BAT Entry number.
Exit	None
Destroys	`A`, `DE`, `HL`,
Calls	F_KRN_MULTIPLY816_SLOW F_KRN_DZFS_SECTOR_TO_CF

4.2.8 F_KRN_DZFS_CHGATTR_FILE

Action	Changes the attributes (RHSE) of a file.
Entry	`DE` = BAT Entry number. `A` = attributes mask byte.
Exit	None
Destroys	`DE`, `HL`,
Calls	F_KRN_MULTIPLY816_SLOW F_KRN_DZFS_SECTOR_TO_CF

4.2.9 F_KRN_DZFS_RENAME_FILE

Action	Changes the name of a file.
Entry	`IY` = **MEMORY** address where the new filename is stored. `DE` = BAT Entry number.
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`, `IY`
Calls	F_KRN_MULTIPLY816_SLOW F_KRN_DZFS_SECTOR_TO_CF

4.2.10 F_KRN_DZFS_FORMAT_CF

Action	Formats a **DISK** with DZFS.
Entry	`HL` = **MEMORY** address where the disk label is stored. `DE` = **MEMORY** address where the number of partitions is stored.
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`, `IX`, `IY`, `tmp_addr1`
Calls	F_KRN_SERIAL_WRSTR F_KRN_DZFS_CALC_SN F_BIOS_RTC_GET_DATE F_BIOS_RTC_GET_TIME F_KRN_BCD_TO_ASCII F_KRN_BIN_TO_BCD4 F_KRN_BIN_TO_BCD6 F_KRN_DZFS_SECTOR_TO_CF F_KRN_SETMEMRNG F_BIOS_SERIAL_CONOUT_A

4.2.11 F_KRN_DZFS_CALC_SN

Action	Calculates the Serial Number (4 bytes) for a **DISK**.
Entry	`IX` = **MEMORY** address where the serial number will be stored.
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`, `IX`
Calls	F_BIOS_RTC_GET_DATE F_BIOS_RTC_GET_TIME F_KRN_MULTIPLY816_SLOW

4.2.12 F_KRN_DZFS_SECTOR_TO_CF

Action	Calls the **BIOS** subroutine that will store the data (512 bytes) currently in CF Card Buffer in **MEMORY**, to the **DISK**.
Entry	`CF_cur_sector` = the sector number in the **DISK** that will be written.
Exit	None
Destroys	`BC`, `DE`
Calls	F_BIOS_CF_WRITE_SEC

4.2.13 F_KRN_DZFS_GET_BAT_FREE_ENTRY

Action	Get number of available BAT entry.
Entry	None
Exit	`CF_cur_file_entry_number` = entry number.
Destroys	`A`, `IY`, `CF_cur_sector`, `CF_cur_file_entry_number`
Calls	F_KRN_DZFS_READ_BAT_SECTOR F_KRN_DZFS_BATENTRY_TO_BUFFER

4.2.14 F_KRN_DZFS_ADD_BAT_ENTRY

Action	Adds a BAT entry into the **DISK**.
Entry	`DE` = BAT entry number. `CF_cur_sector` = Sector number where the BAT Entry is in the **DISK**. `CF Buffer` = Sector (512 bytes) containing the BAT where the entry is. `CF BAT` = BAT Entry data that will be saved to **DISK**.
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`
Calls	F_KRN_MULTIPLY816_SLOW

4.2.15 F_KRN_DZFS_CREATE_NEW_FILE

Action	Creates a new file (and its corresponding BAT Entry) in the **DISK**, from bytes stored in **MEMORY**.
Entry	`HL` = **MEMORY** address of the first byte to be stored. `BC` = number of bytes to be stored in the **DISK**. `IX` = **MEMORY** address where the filename is stored.
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`, `IX`, `tmp_addr1`, `tmp_addr2`, `tmp_addr3`, `tmp_byte`
Calls	F_KRN_DZFS_GET_BAT_FREE_ENTRY F_KRN_DIV1616 F_KRN_MULTIPLY1616 F_KRN_COPYMEM512 F_KRN_CLEAR_CFBUFFER F_KRN_DZFS_SECTOR_TO_CF F_KRN_DZFS_CALC_FILETIME F_KRN_DZFS_CALC_FILEDATE F_KRN_DZFS_SEC_TO_BUFFER F_KRN_DZFS_ADD_BAT_ENTRY F_KRN_DZFS_SECTOR_TO_CF

4.2.16 F_KRN_DZFS_CALC_FILETIME

Action	Packs current Real-Time Clock time into two bytes, which is the format used to store times (created/modified) for files in the **DISK**. The formula used is: $2048 * hours + 32 * minutes + seconds/2$
Entry	None
Exit	`HL` = RTC time
Destroys	`A`, `DE`, `HL`
Calls	F_BIOS_RTC_GET_TIME

4.2.17 F_KRN_DZFS_CALC_FILEDATE

Action	Packs current Real-Time Clock date into two bytes, which is the format used to store dates (created/modified) for files in the **DISK**. The formula used is: $512 * (year - 2000) + month * 32 + day$
Entry	None
Exit	`HL` = RTC date
Destroys	`A`, `DE`, `HL`
Calls	F_BIOS_RTC_GET_DATE

4.2.18 F_KRN_DZFS_SHOW_DISKINFO_SHORT

Action	Outputs to the **CONSOLE** some information of the **DISK**: volume label, serial number, date/time creation.
Entry	None
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`
Calls	F_KRN_SERIAL_WRSTRCLR F_KRN_SERIAL_PRN_BYTE F_KRN_SERIAL_PRN_BYTES F_BIOS_SERIAL_CONOUT_A F_KRN_SERIAL_EMPTYLINES

4.2.19 F_KRN_DZFS_SHOW_DISKINFO

Action	Outputs to the **CONSOLE** all information of the **DISK**: volume label, serial number, date/time creation, file system ID, number of partitions, number of bytes per sector, number of sectors per block.
Entry	None
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`
Calls	F_KRN_SERIAL_WRSTRCLR F_KRN_SERIAL_PRN_BYTE F_KRN_SERIAL_PRN_BYTES F_BIOS_SERIAL_CONOUT_A F_KRN_SERIAL_EMPTYLINES

4.2.20 F_KRN_DZFS_CHECK_FILE_EXISTS

Action	Checks if a specified filename exsists in the **DISK**.
Entry	`HL` = **MEMORY** address where the filename to check is stored.
Exit	`Z Flag` set if filename is not found.
Destroys	`A`, `DE`, `tmp_addr3`
Calls	F_KRN_DZFS_GET_FILE_BATENTRY

4.3 Math Routines

4.3.1 F_KRN_MULTIPLY816_SLOW

Action	Multiplies an 8-bit number by a 16-bit number (HL = A * DE). It does a slow multiplication by adding the multiplier to itself as many times as multiplicand (e.g. 8 * 4 = 8+8+8+8).
Entry	`A` = Multiplicand `DE` = Multiplier
Exit	`HL` = Product
Destroys	`B`, `HL`
Calls	None

4.3.2 F_KRN_MULTIPLY1616

Action	Multiplies two 16-bit numbers (HL = HL * DE)
Entry	`HL` = Multiplicand `DE` = Multiplier
Exit	`HL` = Product
Destroys	`A`, `BC`, `DE`, `HL`
Calls	None

4.3.3 F_KRN_DIV1616

Action	Divides two 16-bit numbers (BC = BC / DE, HL = remainder)
Entry	`BC` = Dividend `DE` = Divisor
Exit	`BC` = Quotient `HL` = Remainder
Destroys	`A`, `BC`, `HL`
Calls	None

4.3.4 F_KRN_CRC16_INI

Action	Initialises the CRC to 0 and the polynomial to the appropriate bit pattern, to generate a CRC-16/BUYPASS1[2].
Entry	None
Exit	`MATH_CRC` = 0 (initial CRC value) `MATH_polynomial` = CRC polynomial
Destroys	`HL`
Calls	None

4.3.5 F_KRN_CRC16_GEN

Action	Combines the previous CRC with the CRC generated from the current data byte, to generate a CRC-16/BUYPASS1[3].
Entry	`A` = current data byte. `MATH_CRC` = previous CRC `MATH_polynomial` = CRC polynomial
Exit	`MATH_CRC` = CRC with current data byte included
Destroys	`A`, `BC`, `DE`, `HL`
Calls	None

4.4 String manipulation Routines

4.4.1 F_KRN_IS_PRINTABLE

Action	Checks if a character is a printable ASCII character.
Entry	`A` = character to check.
Exit	`C Flag` is set if character is printable.
Destroys	None
Calls	None

4.4.2 F_KRN_IS_NUMERIC

Action	Checks if a character is numeric (0, 1, 2, 3, 4, 5, 6, 7, 8 or 9).
Entry	`A` = character to check.
Exit	`C Flag` is set if character is numeric.
Destroys	None
Calls	None

4.4.3 F_KRN_TOUPPER

Action	Converts a charcater to uppercase (e.g. *a* is converted to `A`).
Entry	`A` = character to convert.
Exit	`A` = uppercased character.
Destroys	None
Calls	None

4.4.4 F_KRN_STRCMP

Action	Compares two strings.
Entry	`A` = length of string 1. `HL` = **MEMORY** address where the first byte of string 1 is located. `B` = length of string 2. `DE` = **MEMORY** address where the first byte of string 2 is located.
Exit	if str1 = str 2, `Z Flag` set and `C Flag` not set. if str1 != str 2 and str1 longer than str2, `Z Flag` not set and `C Flag` not set. if str1 != str 2 and str1 shorter than str2, `Z Flag` not set and `C Flag` set.
Destroys	`A`, `BC`, `DE`,`HL`
Calls	None

4.4.5 F_KRN_STRCPY

Action	Copies *n* characters from string 1 to string 2.
Entry	`HL` = **MEMORY** address where the first byte of string 1 is located. `DE` = **MEMORY** address where the first byte of string 2 is located. `B` = number of characters to copy.
Exit	None
Destroys	`A`, `DE`, `HL`
Calls	None

4.4.6 F_KRN_STRLEN

Action	Gets the length of a string that is terminated with a specified character.
Entry	`HL` = **MEMORY** address where the first byte of the string is located. `A` = terminating character.
Exit	`B` = lenght of the string.
Destroys	`BC`, `HL`
Calls	None

4.5 Conversion Routines

4.5.1 F_KRN_ASCIIADR_TO_HEX

Action	Convert an address (or any 2 bytes) from hex ASCII to its hexadecimal value (e.g. 32 35 37 30 are converted into 2570).
Entry	`IX` = **MEMORY** address where the first byte is located.
Exit	`HL` = hexadecimal converted value.
Destroys	`HL`
Calls	F_KRN_ASCII_TO_HEX

4.5.2 F_KRN_ASCII_TO_HEX

Action	Converts two ASCII characters (representing two hexadecimal digits) ; to one byte in hexadecimal (e.g. 0x33 and 0x45 are converted into 3E).
Entry	`H` = Most significant ASCII digit. `L` = Less significant ASCII digit.
Exit	`A` = Converted value.
Destroys	`A`, `BC`
Calls	None

4.5.3 F_KRN_HEX_TO_ASCII

Action	Converts one byte in hexadecimal to two ASCII printable characters (e.g. 0x3E is converted into 33 and 45, which are the ASCII values of 3 and E).
Entry	`A` = Byte to convert.
Exit	`H` = Most significant ASCII digit. `L` = Less significant ASCII digit.
Destroys	`A`, `BC`, `HL`
Calls	None

4.5.4 F_KRN_BIN_TO_BCD4

Action	Converts a byte of unsigned integer hexadecimal to 4-digit BCD (e.g. 0x80 is converted into 0128).
Entry	`A` = Unsigned integer to convert.
Exit	`H` = Hundreds digits. `L` = Tens digits.
Destroys	`A`, `BC`, `HL`
Calls	None

4.5.5 F_KRN_BIN_TO_BCD6

Action	Converts two bytes of unsigned integer hexadecimal to 6-digit BCD (e.g. 0xFFFF is converted into 065535).
Entry	`HL` = Unsigned integer to convert.
Exit	`C` = Thousands digits. `D` = Hundreds digits. `E` = Tens digits.
Destroys	`A`, `BC`, `DE`, `HL`
Calls	None

4.5.6 F_KRN_BCD_TO_ASCII

Action	Converts 6-digit BCD to hexadecimal ASCII string (e.g. 512 is converted into 30 30 30 35 31 32).
Entry	`DE` = **MEMORY** address where the converted string will be stored. `C` = first two digits of the 6-digit BCD to convert. `H` = next two digits of the 6-digit BCD to convert. `L` = last two digits of the 6-digit BCD to convert.
Exit	None
Destroys	`A`, `DE`
Calls	None

4.5.7 F_KRN_BITEXTRACT

Action	Extracts a group of bits from a byte and returns the group in the LSB position.
Entry	`E` = byte from where to extract bits. `D` = number of bits to extract. `A` = start extraction at bit number.
Exit	`A` = extracted group of bits
Destroys	`A`, `BC`, `DE`, `HL`
Calls	None

4.5.8 F_KRN_BIN_TO_ASCII

Action	Converts a 16-bit signed binary number (-32768 to 32767) to ASCII data (e.g. 32767 is converted into 33 32 37 36 37).
Entry	`D` = High byte of value to convert. `E` = Low byte of value to convert.
Exit	`CLI_buffer_pgm` = converted ASCII data. First byte us the length.
Destroys	`A`, `BC`, `DE`, `HL`, `CLI_buffer_pgm`
Calls	None

4.5.9 F_KRN_DEC_TO_BIN

Action	Converts an ASCII string consisting of the length of the number (in bytes), a possible ASCII - or + sign, and a series of ASCII digits to two bytes of binary data. Note that the length is an ordinary binary number, not an ASCII number. (e.g. 33 32 37 36 37 is converted into 7FFF).
Entry	`HL` = **MEMORY** address where the string to be converted is.
Exit	`HL` = converted bytes.
Destroys	`A`, `BC`, `DE`, `HL`, `tmp_byte`
Calls	None

4.5.10 F_KRN_PKEDDATE_TO_DMY

Action	Extracts day, month and year from a packed date (used by DZFS to store dates).
Entry	`HL` = packed date.
Exit	`A` = day. `B` = month. `C` = year.
Destroys	`A`, `BC`, `HL`, `tmp_addr1`
Calls	None

4.5.11 F_KRN_PKEDTIME_TO_HMS

Action	Extracts hour, minutes and seconds from a packed time (used by DZFS to store times).
Entry	`HL` = packed time.
Exit	`A` = hour. `B` = minutes. `C` = seconds.
Destroys	`A`, `BC`, `HL`, `tmp_addr1`
Calls	None

4.6 MEMORY Routines

4.6.1 F_KRN_SETMEMRNG

Action	Sets (changes) a value in a **MEMORY** position range.
Entry	`HL` = **MEMORY** start position (first byte). `BC` = number of bytes to set. `A` = value to set.
Exit	None
Destroys	`BC`, `HL`
Calls	None

4.6.2 F_KRN_COPYMEM512

Action	Copies bytes from one area of **MEMORY** to another, in group of 512 bytes (i.e. max. 512 bytes). If less than 512 bytes are to be copied, the rest will be filled with zeros.
Entry	`HL` = **MEMORY** origin position (from where to copy the bytes). `DE` = **MEMORY** destination position (to where to copy the bytes). `BC` = number of bytes to copy (MUST be less or equal to 512).
Exit	None
Destroys	`A`, `BC`, `DE`, `HL`
Calls	None

4.6.3 F_KRN_SHIFT_BYTES_BY1

Action	Moves bytes (by one) to the right and replaces first byte with bytes counter.
Entry	`HL` = **MEMORY** address of last byte to move. `BC` = number of bytes to move.
Exit	None
Destroys	`A`, `DE`, `HL`
Calls	None

4.6.4 F_KRN_CLEAR_MEMAREA

Action	Clears (with zeros) a number of bytes, starting at a specified **MEMORY** address. Maximum 256 bytes can be cleared.
Entry	`IX` = **MEMORY** address of first byte to clear. `B` = number of bytes to clear.
Exit	None
Destroys	`A`, `BC`, `IX`
Calls	None

4.6.5 F_KRN_CLEAR_CFBUFFER

Action	Clears (with zeros) the **MEMORY** area of the **DISK** buffer.
Entry	None
Exit	None
Destroys	`BC`, `IX`
Calls	F_KRN_CLEAR_MEMAREA

5 dastaZ80 File System (DZFS)

In summary, a file system is a layer of abstraction to store, retrieve and update a set of files.

A file system manages access to the data and the metadata of the files, and manages 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.

DZFS main goal is to be very simple to implement. As the free **MEMORY** (i.e. **RAM** - OS - System variables and buffers) of the dastaZ80 is about 55,952 bytes, it makes no sense to have files bigger than that, as will not fit. Therefore, DZFS defines that *a Block can store only a single file*.

dastaZ80 access the **DISK** via Logical Block Addressing (LBA), which is a particularly simple linear addressing schema, in which each sector is assigned a unique number rather than referring to a cylinder, head, and sector (CHS) to access the disk.

A typical LBA scheme uses a 28-bit value that allows up to 8.4 GB of data storage capacity. DZFS schema is as follows:

LBA 3	LBA 2	LBA 1	LBA 0
0000	0000 00PP	BBBB BBBB	BBSS SSSS

Where:

- S is Sector (6 bits)
- B is Block (10 bits)
- P is Partition (2 bits)
- 0 not used (10 bits)

5.1 DZFS characteristics

- **Bytes per Sector**: 512
- **Sectors per Block**: 64
- **Bytes per Block**: 32,768 (64 * 512). This also defines the maximum size of a file and the BAT maximum size.
- **Bytes per BAT entry**: 32
- **BAT entries**: 1024 (32,768 / 32). This also defines the maximum number of files per Partition.
- **Blocks per Partition**: 1,024 (1 reserved for BAT)
- **Sectors per Partition**: 65,536 (1,024 * 64)

- **Bytes per Partition**: 33,587,200 (1,024 * 32,768 + 1 BAT Block)
- **Partitions per Disk**: 3 (125 MB1) / 33,587,200)

5.2 DISK anatomy

A disk (128 MB CompactFlash in our case) is divided into areas:

- **Superblock** = 512 bytes
- **Partition 1**
  - **Block Allocation Table (BAT)** = 1 Block
  - **Data Area** = 1023 Blocks
- **Partition 2**
  - **Block Allocation Table (BAT)** = 1 Block
  - **Data Area** = 1023 Blocks
- **Partition 3**
  - **Block Allocation Table (BAT)** = 1 Block
  - **Data Area** = 1023 Blocks

5.2.1 Superblock

The first 512 bytes on the **DISK** contain fundamental information about the 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* or *MBR* for short. In DZFS, it is called *Superblock*, as it is an orphan sector that doesn't belong to any block.

Offset	Length (bytes)	Description	Example
0x00	2	Signature. Used to check that this is a Superblock. Set to 0xABBA	`AB BA`
0x02	1	Not used	`00`
0x03	8	File system identifier. ASCII values for human-readable. Padded with spaces.	`DZFSV1`
0x0B	4	Volume serial number	`35 2A 15 F2`
0x0F	1	Not used.	`00`
0x10	16	Volume Label. ASCII values. Padded with spaces.	`dastaZ80 Main`
0x20	8	Volume Date creation. ASCII values (ddmmyyyy).	`03102022`

Offset	Length (bytes)	Description	Example
0x28	6	Volume Time creation. ASCII values (hhmmss).	`142232`
0x2E	2	Bytes per Sector (in Hexadecimal little-endian)	`00 02`
0x30	1	Sectors per Block (in Hexadecimal)	`40`
0x31	1	Number of Partitions	`01`
0x32 - 0x64	51	Copyright notice (ASCII value)	`Copyright 2022David Asta The MIT License (MIT)`
0x65 - 0x1FF	411	Not used (filled with 0x00)	`00 00 00 00 00 00 00 .........`

5.2.2 Block Allocation Table (BAT)

The BAT is an area of 32 bytes on the **DISK** used to store the details about the files saved in the Data Area, and is comprised of file descriptors called *entry*. Each entry holds information about a single file.

For simplicity, each entry works also as index. The first entry describes the first file on the **DISK**, the second entry describes the second file, and so on.

Offset	Length (bytes)	Description	Example
0x00	14	**Filename** Padded with spaces at the end. (only allowed A to Z and 0 to 9. No spaces allowed. Cannot start with a number.) First character also indicates 00=available, 7E=deleted (will appear as)	`46 49 4C 45 30 30 30 30 31 20 20 20 20 20`
0x0E	14	**Attributes** (0=Inactive / 1=Active) Bit 0 = Read Only Bit 1 = Hidden	Read Only, System file, Executable = 1101 = `0D`

Offset	Length (bytes)	Description	Example
		Bit 2 = System Bit 3 = Executable Bit 4-7 = Not used	
0x0F	2	**Time created** 5 bits for hour (binary number 0-23) 6 bits for minutes (binary number 0-59) 5 bits for seconds (binary number seconds / 2)	`F5 9A`
0x11	2	**Date created** 7 bits for year since 2000 (max. is year 2127) 4 bits for month (binary number 0-12) 5 bits for day (binary number 0-31)	`69 1B`
0x13	2	Time last modified (same formula as Time created)	`F5 9A`
0x15	2	Date last modified (same formula as Date created)	`69 1B`
0x17	2	File size in bytes (little-endian)	`26 00`
0x19	1	File size in sectors (little-endian)	`01`
0x1A	2	Entry number (little-endian)	`00 00`
0x1C	2	**1st Sector** (where the file data starts) It is calculated when the file is created. The formula is: 65 + 64 * entry_number	`41 00`
0x1E	2	Load address (The start address little-endian where it will be loaded in RAM)	`68 25`

5.2.3 Data Area

The Data Area is the area of the **DISK** used to store file data (e.g. programs, documents).

It is divided into Blocks of 64 Sectors each.

6 How To

6.1 Read data from DISK

Given `CF_is_formatted` is equal to 0xFF (i.e. **DISK** is formatted with DZFS file system), call F_KRN_DZFS_LOAD_FILE_TO_RAM with `DE` equal to first sector (512 bytes) to read and `IX` equal to how many sectors to read.

Read bytes will be copied into **MEMORY**, starting at the address equal to the address stored at `CF_cur_file_load_addr` which is stored in the Block Allocation Table (BAT) in **DISK**.

6.2 Write data to DISK

Given `CF_is_formatted` is equal to 0xFF (i.e. **DISK** is formatted with DZFS file system):

- Store the filename (in ASCII) somewhere in `MEMORY`.
- call F_KRN_DZFS_GET_FILE_BATENTRY, with `HL` equal to the `MEMORY` address where the filename is stored. If a file with the specified filename does not exist, flag `z` will be set, therefore it is OK to save the file.
- call F_KRN_DZFS_CREATE_NEW_FILE, with `HL` equal to the address in **MEMORY** of first byte to be stored, `BC` equal to the total number of bytes to be stored, and `IX` equal to the address in **MEMORY** where the filename is stored.

7 Appendixes

7.1 ANSI Terminal colours

- ANSI_COLR_BLK - Black
- ANSI_COLR_RED - Red
- ANSI_COLR_GRN - Green
- ANSI_COLR_YLW - Yellow
- ANSI_COLR_BLU - Blue
- ANSI_COLR_MGT - Magenta
- ANSI_COLR_CYA - Cyan
- ANSI_COLR_WHT -
- ANSI_COLR_GRY - Grey

7.2 How DZFS Volume Serial Number is calculated

Calculated by combining the date and time at the point of format:

- first byte is calculated as follows:
  - day + miliseconds (converted to hexadecimal)
  - e.g. 3 + 50 = 53 (0x35)
- second byte is calculated as follows:
  - month + seconds (converted to hexadecimal)
  - e.g. 10 + 32 = 42 (0x2A)
- last two bytes are calculated as follows:
  - (hours [if pm + 12] * 256) + minutes + year (converted to hexadecimal)
  - e.g. (2 + 12 = 14 * 256 = 3584) + 22 + 2012 = 5618 (0x15 0xF2)

7.3 OS Boot Sequence

After power on or after pressing the **RESET** button:

- **Bootstrap**
  - Copy contents of the ROM into High RAM (0x8000 - 0xFFFF).
  - Disable ROM chip and enable Low RAM (0x0000 - 0x7FFF). Therefore, all **MEMORY** is RAM from now on.

  - Copy the copy of ROM inm High RAM to Low RAM. Bootstrap code is not copied.
  - Transfer control to BIOS (`jp F_BIOS_SERIAL_INIT`)
- **Initialise SIO/2**
  - Initialise SIO/2
    - Set Channel A as 115,000 bps, 8N1, Interrupt in all received characters.
    - Set Channel B as 115,000 bps, 8N1, Interrupt in all received characters.
    - Set Interrupt Vector to 0x60.
  - Set CPU to Interrupt Mode 2.
  - `jp F_BIOS_WBOOT`
- **BIOS Boot**
  - Set SIO/2 Channel A as primary I/O.
  - Transfer control to Kernel ()`jp F_KRN_START`).
- **Kernel Boot**
  - Display dzOS welcome message.
  - Display dzOS release version.
  - Display Kernel version.
  - Display available RAM.
  - Initialise CompactFlash Card.
  - Display volume ID, Serial Number and date/time of format.
  - Detect Real-Time Clock (RTC).
  - Display RTC's battery status.
  - Transfer control to Command-line Interpreter (CLI) ()`jp F_CLI_START`).
- **CLI**
  - Display CLI version.
  - Clear command buffers
  - Display prompt (¿).
  - Read command entered by user.

- Parse command.
- Execute command.
- Loop back to Display prompt.