BBC

DESIGNS DEPARTMENT

DESIGNS DEPARTMENT HANDBOOK

No. 2.467(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 4 - System Software Interfaces

BRITISH BROADCASTING CORPORATION
ENGINEERING DIVISION

DESIGNS DEPARTMENT HANDBOOK

No. 2.467(82)

ZELDA Development System

EP1M/27

Users´ Handbook

Part 4 - System Software Interfaces

....
(D. C. Savage)
for Head of Designs Department

Written by: N. A. F. Cutmore
D. J. King
R. T. Russell

A copy of the master is held on disk file. Any amendments must include revision of both the disk file and the paper master, together with a new issue date.

The revision must be approved by the Head of Monitoring and Control Section.

DESIGNS DEPARTMENT HANDBOOK

No. 2.467(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 4 - System Software Interfaces

CONTENTS

DESIGNS DEPARTMENT HANDBOOK

No. 2.467(82)

ZELDA Development System

EP1M/27

Users´ Handbook

Part 4 - System Software Interfaces

1. Introduction

ZELDA (Zeus Editor, Loader, Disk operating system and Assembler) is a Z80-based software development system built from Zeus modules. The system is a microcomputer with twin 8-inch floppy disks, 49 Kbytes of memory, keyboard, VDU and printer. It may be used for a variety of applications other than Z80 assembly language software development by using programs supplied on disk with the system.

2. Documentation

The Zelda System Users´ Handbook is divided into four parts. These are:

Part 1 - Hardware 2.464(82)

Describes the configuration of the sub-units, user I/O connections, how to expand the system, and is supplied with the Handbooks of all sub-units.

Part 2 - Firmware 2.465(82)

Describes the resident Monitor, Text Editor, Z80 Mnemonic Assembler, Relocating Linking Loader and Peripheral Interchange Program, and explains how to use them.

Part 3 - System Utilities 2.466(82)

Describes the utility programs supplied with the system on floppy disk.

Part 4 - System Software Interfaces (this document)

Describes in detail the resident routines available to the user, and illustrates how to write programs for use on ZELDA.

Other relevant documents include:

Zeus System EDI	EDI 10412
Zeus Users' Manual	DDTM
A Modular 8-bit Microcomputer	DDTM 2.447(80)
Automatic Fault Detection	DDTM 2.448(80)
Equipment with Customer Options	DDTM 2.449(80)
BBC Code of Practice in the use of PROMs	DDTM 1.155(80)

3. Foreword

This document provides a brief description of useful subroutines in the Zelda firmware which may be called by the user. Future issues of the firmware will preserve the addresses and register usage given.

The reader is expected to have read and be familiar with the terms used in previous parts of the Zelda System Users' Handbook (D.D.H.B. 2.464-6).

Under certain error conditions, several of the Disk Operating System (DOS) routines listed in section 5 of this handbook may transfer command to the PIP program (see section 5 of Part 2). To save space, each of the possible errors has been given a number as follows :-

1) FILE NOT FOUND
2) FILE ALREADY EXISTS
3) ILLEGAL FILENAME
4) DISK FULL
5) DIRECTORY FULL
6) DIRECTORY ERROR
7) ILLEGAL TRACK NUMBER
8) DEVICE DOES NOT EXIST
9) NO OPEN FILE ON CHANNEL n
10) DISK FAIL -

A fuller description of the meaning of these error messages can be found in Appendix 1 of part 2 of the handbook.

4. ROUTINES WITHIN THE SYSTEM MONITOR

These routines are subroutines callable by the user except where stated otherwise.

4.1

NAME: RENTRY
ADDRESS: E11D
FUNCTION: ENTER MONITOR PROGRAM

This address provides the means for a user program to pass control back to the system monitor. A jump or call to this address will transfer command to the monitor program as detailed in section 4 of part 2 (D.D.H.B. 2.465(82)). Any disk files left open when this address is accessed will be closed and, if a disk error occurs, control will pass to the Peripheral Interchange Program as detailed in section 5 of part 2. No inputs are required but since the routine unconditionally sets interrupts to Mode 2, the user must ensure that any interrupts left enabled will work in this mode.

4.2

NAME: PTXT
ADDRESS: E3C7
FUNCTION: PRINT TEXT

This routine sends a text string to the specified output channel and then returns to the calling program. Immediate return mode (bit 7 of E = 1) is not permitted when using this routine.

On input — Bits 0,1 and 2 of E determine channel in use (001, 011 or 101)
Bit 3 of E if set will initialise the driver (see WRCHR (4.5))
HL points to the first character of the string to be output (terminated by ETX)

On output — A and D contain ETX
Bits 3 and 7 of E are reset by the driver in use
HL addresses the ETX
The zero flag is set

Destroys — Registers A,D,E,H,L and flags

This routine calls WRCHR (4.5).

4.3

NAME: SCAN
ADDRESS: E414
FUNCTION: INPUT OPERANDS

This routine may be used to obtain from the operator up to three sixteen bit values for use in a program. The numbers entered are stored in specific places in RAM along with a record of how many values there are and how the routine was exited.

This routine has no input requirements.

On output OPR1 contains the 1st operand entered
OPR2 contains the 2nd operand entered
OPR3 contains the 3rd operand entered
OPFLG contains the number of operands returned (max 3)
NXTCHR contains the terminator entered

Destroys All registers (including the alternate set) except SP and I

See section 6 for a list of the RAM addresses allocated to the labels.

The routine requires the operator to type in up to three operands and will return immediately if a non-allowed character is entered. A description of what constitutes a legal operand is given in section 4.6 of part 2 of this handbook. The only acceptable terminators are 0D Hex and 5E Hex (RETURN and CARAT); any other terminator will cause an error in the value stored in OPFLG. The user is advised to check the contents of NXTCHR on return from this routine and should not use the values returned in OPR1-3 if an illegal terminator is found.

If more than three operands are entered, the third and subsequent ones will be summed and placed in OPR3; OPFLG will be set to 3.

See section 8.2 in this document for an example of the use of the SCAN routine.

4.4

NAME: RDCHR
ADDRESS: E522
FUNCTION: READ CHARACTER

This routine will call the input driver allocated to the channel determined by bits 1 & 2 of the E register. This is the normal way for a user program to read from a disk file or to get entries from the keyboard, etc.

On input — Bits 0, 1 and 2 of E determine channel in use:
000 will use :CI
010 will use :OI
100 will use :SI
Bit 3 of E causes driver initialisation if set (and a "rewind" operation if this facility is supported by the driver).
Bit 7 of E selects immediate return mode if set (if supported by the driver).

On output — If bit 7 of E is reset, A and D contain the character read.
If bit 7 of E is set an immediate return took place.
Bit 3 of E is reset (by the driver).
The carry flag is set on physical EOF (disk files only).

Destroys — Registers A,D,E and flags

Note that the disk file structure of Zelda requires use of the carry flag to determine when the end of a binary file is reached. Any modifications the user may make to a driver that reads disk files should ensure that the carry flag is not destroyed.

See Appendix 4 in part 2 for details of how to write a new driver routine, and for a description of the operation of the initialise and immediate return functions.

To read a disk file it is necessary to use ASSIGN (5.5) or GETDF (5.6) before using this routine - (see section 8.1 of this document for an example).

4.5

NAME: WRCHR
ADDRESS: E527
FUNCTION: WRITE CHARACTER

This routine will call the output driver allocated to the channel determined by bits 1 & 2 of the E register. This is the normal way for a user program to write to a device/disk file or to the console.

On input — D contains the character to be written.
Bits 0, 1 & 2 of E determine channel in use:
001 will use :CO
011 will use :OO
101 will use :SO
Bit 3 of E causes driver initialisation if set.
Bit 7 of E selects immediate return mode if set (if supported by the driver).

On output — If bit 7 of E is reset, A and D contain the character written.
If bit 7 of E is set an immediate return took place.

Bit 3 of E is reset (by the driver).

Destroys   Registers A,E and flags

See Appendix 4 in part 2 for details of how to write a new driver routine, and for a description of the operation of the initialise and immediate return functions.

To write to a disk file it is necessary to use ASSIGN (5.5) or GETDF (5.6) before using this routine - (see section 8.1 of this document for an example).

4.6

NAME: SRCHR
ADDRESS: E541
FUNCTION: SEARCH RESIDENT MNEMONICS

This routine examines the resident mnemonic list held in PROM to see if a specified driver, jump address or register mnemonic exists.

On input   HL contains mnemonic (e.g. H = "T" and L = "P" will search for :PT)

On output   HL contains the mnemonic value (i.e. where to look for data or where to call) if the mnemonic is found, otherwise HL is unchanged
The zero flag is set if the mnemonic is found, reset if not

Destroys   Registers A,H,L and flags

This routine is normally used with its companion SRCHU (4.7) which examines the RAM mnemonic list. The following mnemonics are stored in the PROM table :-
:A, :F, :H, :L, :B, :C, :D, :E, :IX, :IY, :I, :IF, :SP, :PC , :CI, :CO, :OI, :OO, :SI, :SO, :TK, :TT and :ER.

(See section 4.6 of part 2 for further information).

All other mnemonics (if they exist) will be in the RAM table. Note that a single letter mnemonic (e.g :A) is still considered to be two characters long when searching - in this case, the H register should contain 20 Hex (a space).

4.7

NAME: SRCHU
ADDRESS: E547
FUNCTION: SEARCH USER MNEMONICS

This routine performs exactly the same function as SRCHR (4.6) but examines the user's mnemonic list held in RAM (see section 4.6 of part 2 for further details).

Inputs and outputs are as for SRCHR.

4.8

NAME: ASBIN
ADDRESS: E583
FUNCTION: CONVERT ASCII TO BINARY

This routine will convert a single ASCII hexadecimal character into its binary value.

On input A contains the character to be converted

On output A contains the binary value associated with the character

Destroys Register A and flags

No check is made of the Hex character on entry. If it is in the range "0" - "9", the routine subtracts 30 Hex; for any other character it subtracts 37 Hex.

4.9

NAME: PACC
ADDRESS: E58B
FUNCTION: PRINT ACCUMULATOR (HEX)

This routine sends to the specified output channel a two character hexadecimal representation of the value in the accumulator. Immediate return mode (specified by bit 7 of E = 1) is not permitted when using this routine.

On input Bits 0, 1 & 2 of E determine the channel in use
Bit 3 of E if set will initialise the driver (see WRCHR (4.5))

On output A and D contain the second digit printed
Bits 3 and 7 of E are affected by the driver in use

Destroys Registers A,D,E and flags

4.10

NAME: ECHO
ADDRESS: E597
FUNCTION: READ THEN WRITE A CHARACTER

This routine provides a single call to read and write a character. A call to this routine is equivalent to :-

```
CALL RDCHR
CALL WRCHR
```

Immediate return mode (bit 7 of E = 1) is not permitted.
See entries for RDCHR (4.4) and WRCHR (4.5).

4.11

NAME: CRLF
ADDRESS: E59C
FUNCTION: SEND A RETURN THEN A LINE FEED

This routine sends to the specified output channel a carriage return (0D Hex), followed by a line feed (0A Hex). It then returns to the calling program. Immediate return mode (specified by bit 7 of E = 1) is not permitted when using this routine.

On input	Bits 0, 1 & 2 of E determine the channel in use Bit 3 of E if set will initialise the driver (see WRCHR (4.5))
On output	A and D contain line feed (0A Hex) Bits 3 and 7 of E are affected by the driver in use
Destroys	Registers A,D,E and flags

4.12

NAME: SPACE
ADDRESS: E5A5
FUNCTION: PRINT A SPACE

This routine sends to the specified output channel a space (20 Hex) and then returns to the calling program.

A call to this routine is equivalent to :-

```
LD   D,' '
CALL WRCHR
```

See entry for WRCHR (4.5).

4.13

NAME: PASP
ADDRESS: E5AA
FUNCTION: PRINT ACCUMULATOR THEN SPACE

A call to this routine is equivalent to :-

```
CALL PACC
CALL SPACE
```

See entries for PACC (4.9) and SPACE (4.12).

4.14

NAME: PRVAL
ADDRESS: E5AF
FUNCTION: PRINT HEX DIGIT

This routine sends the specified output channel a single character corresponding to the hexadecimal value of the four least significant bits of the accumulator.

On input Bits 0 - 3 of A contain value to be converted
Bits 0, 1 & 2 of E determine the channel in use
Bits 3,7 of E specify initialise/immediate return (see WRCHR (4.5))

On output A and D contain the character sent
Bits 3 and 7 of E are affected by the driver in use

Destroys Registers A,D,E and flags

4.15

NAME: PADDO
ADDRESS: E604
FUNCTION: PRINT ADDRESS AND SPACE

This routine sends to the specified output device a four character hexadecimal representation of the value in the HL register pair, followed by a space. It then returns to calling program.

A call to this routine is equivalent to :-

```
LD   A,H
CALL PACC
LD   A,L
CALL PACC
CALL SPACE
```

See entries for PACC (4.9), SPACE (4.12) and WRCHR (4.5).

4.16

NAME: TTID (mnemonic TK:)
ADDRESS: E689
FUNCTION: KEYBOARD INPUT DRIVER

This routine examines the 8251 USART servicing the Zelda keyboard. If a character is waiting, it is returned with bit 7 reset. If no character is available, and the immediate return flag (bit 7 of E) is reset, the routine will loop until the character is entered.

On input Bit 7 of E if set will force immediate return if no character is waiting.

On output If bit 7 of E is reset on output then A and D contain the character read with bit 7 reset.

If bit 7 of E is set on output then A is destroyed

Destroys Registers A,D,E and flags

The normal console input driver is a routine which performs "timeout" and "not ready" detection on the disk drives before passing control to TTID. The user should be aware that it is possible to corrupt disks by writing the wrong directory onto them if the console input vector is altered (as a disk change will not be noticed).

4.17

NAME: VDUOUT (mnemonic TT:)
ADDRESS: E6F9
FUNCTION: VDU OUTPUT DRIVER

This routine takes the character in the D register and puts it onto the VDU screen at the current cursor position unless it is a control code. See Appendix 5 of part 2 for a description of how control codes are interpreted by this routine.

On input D contains the character to be written
Bit 3 of E if set will initialise the driver

On output A and D contain the character
Bits 3 and 7 of E are reset

If bit 3 of E is set on input, the driver is arranged so that subsequent characters will be printed in normal video and control codes will perform their usual functions (i.e. the effects of ESCAPE or SHIFT OUT are negated). See Appendix 5 of part 2 of this handbook for details of the effects of these and other control codes.

The normal console output driver is a routine which performs extra functions before passing control to VDUOUT (see Appendix 5 part 2). These functions are important for correct operation of a large part of the system and the console output channel should not normally be reallocated to a different driver.

4.18

NAME: MOVCSR
ADDRESS: E774
FUNCTION: MOVE CURSOR

This routine moves the cursor on the VDU screen by a specified number of character places

On input A contains the number of places required (in the range - 80 to + 80 in signed integer format - i.e. B0 to 50 Hex)

On output The cursor is set to the new position and the screen is scrolled if needed

Destroys   Register A and flags

If the cursor is moved downwards off the bottom line, or off the right end of the bottom line, the screen will scroll up by one line. If the cursor is moved upwards off the top line, it will wrap around onto the bottom line and the screen will not scroll.

4.19

NAME: GETCSR
ADDRESS: E7A7
FUNCTION: GET CURSOR ADDRESS

This routine returns with the HL register pair containing the current VDU RAM address where the cursor is positioned. The value returned will be in the range A000 - A7FF Hex.

On output   HL contains the current screen address of the cursor

Destroys   H,L

4.20

NAME: OFFSET
ADDRESS: E7F1
FUNCTION: RETURN RELATIVE CURSOR ADDRESS

This routine returns with the HL register pair containing the number of character positions between the top left-hand corner of the screen and the current cursor position. The user can calculate from this value the row and column the cursor is on.

On output   DE contains the current address of the top left-hand corner of the screen
HL contains the number of character positions between the top left-hand corner and the current cursor position

Destroys   Registers A,D,E,H,L and flags

5. ROUTINES WITHIN THE DISK OPERATING SYSTEM

These routines are subroutines callable by the user except where stated otherwise.

5.1

NAME: DSKIN
ADDRESS: E800
FUNCTION: DISK INPUT DRIVER

This routine reads a byte from a disk file or several concatenated files. It requires a parameter block in RAM which is usually set up by a call to the ASSIGN routine. See entries for ASSIGN (5.5) and DSKIO (5.25).

On input: Bit 1 of E determines which of two parameter blocks will be used. If 0, the "source input" parameter block will be used and if 1 the "object input" block will be used (corresponding usually to channels 4 and 2 respectively).
Bit 3 of E if set will force the routine to go back to the start of the first file in the file list ("rewind")

On output: A and D contain the character read or ETX if the end of the file is reached
Bits 0,3 and 7 of E are reset
The carry flag is set if the physical end of the file is reached (the last byte in the last sector has been read)

Destroys: Registers A,D,E and flags

This routine will abort to PIP if disk error 1,6,7,8 or 10 occurs. (See section 3 for an explanation of these numbers). It may also enable interrupts.

5.2

NAME: DSKOUT
ADDRESS: E803
FUNCTION: DISK OUTPUT DRIVER

This routine writes a byte to a disk file. It requires a parameter block in RAM which is usually set up by a call to the ASSIGN routine. See entries for ASSIGN (5.5) and DSKOO (5.26).

On input: D contains the byte to be written
Bit 1 of E determines which of two parameter blocks will be used. If 0, the "source output" parameter block will be used and if 1 the "object output" block will be used (corresponding usually to channels 5 and 3 respectively).
Bit 3 of E if set will open the file (unless already open)

On output A and D contain the character sent
Bit 0 of E is set
Bits 3 and 7 of E are reset

Destroys Registers A,E and flags

This routine will abort to PIP if disk error 2,3,4,5,6,7,9 or 10 occurs. (See section 3 for an explanation of these numbers). It may also enable interrupts.

5.3

NAME: CLOSE
ADDRESS: E806
FUNCTION: CLOSE OUTPUT DISK FILE

This routine fills the remainder of the last buffered sector with NUL characters. It then stores the rest of the file still in the RAM buffer on the disk and updates the disk directory to include the file.

On input Bits 0,1 and 2 of E determine the channel in use

Destroys Registers A,B,C,D,E,H,L,IX and flags

This routine will abort to PIP if disk error 5,6 or 10 occurs. (See section 3 for an explanation of these numbers).

5.4

NAME: PIP
ADDRESS: E809
FUNCTION: PERIPHERAL INTERCHANGE PROGRAM

This routine allows manipulation of disk files, e.g. deleting and renaming files, and provides a general means of transferring data between disk files or drivers. It can also be used to list an index of disk contents.

A jump or call to this routine has the same effect as E :PI executed from within the monitor. See section 5 of part 2 of this handbook and the entry for PIPEXT (5.15) in this section for further details.

This routine will NOT return to the user program. See the entry for PIPEXT (5.15) for a method of accessing the commands of PIP without losing control.

5.5

NAME: ASSIGN
ADDRESS: E80C
FUNCTION: ASSIGN A CHANNEL TO A DEVICE OR DISK FILE

This routine is called to allocate an I/O channel to a device or file whose name is held in RAM. It sets up the parameter block required by many other routines e.g. DSKIN or DSKOUT. See entries for DSKI0 (5.25) and DSKO0 (5.26) for further information.

If no device mnemonic is specified, the routine assumes a disk 0 filename is being used. Both resident and user mnemonics are recognised.

On input — D contains the default file extension that will be used if none is specified
Bits 0,1 and 2 of E determine channel in use
HL points to the first character of the device or filename string terminated by carriage return. If this string describes an input disk file, it must be preserved until the file is no longer required (unless DSKI0 (5.25) is being used in which case only the first file will be accessed and the string need not be stored).

On output — If the carry flag is reset, the RAM string on input was blank
If the carry is set:
DE contains the driver routine address allocated
HL points to the device/filename delimeter in the string
IX points to the parameter block allocated to the channel
The zero flag is set if a disk file was specified
The RAM vector for the I/O channel is set (see section 4.3 of part 2)

Destroys — Registers A,D,E,H,L,IX and flags

This routine will abort to PIP if a specified device does not exist.

5.6

NAME: GETDF
ADDRESS: E80F
FUNCTION: GET A DEVICE/FILENAME

This routine sends a text string to the console output channel and waits for the operator to type in a filename and/or device mnemonic. When the RETURN key is pressed, control passes to the ASSIGN routine above.

On input — D contains the default extension to be used if none is specified by the operator
Bits 0,1 and 2 of E determine the channel in use

HL points to the text string to be output (terminated by an ETX)

The outputs from this routine are the same as for ASSIGN (5.5).

If the operator presses the ESCAPE key, control will pass to the monitor and the routine will not return to the calling program.

This routine calls PTXT (4.2) and STRING (6.1).

5.7

NAME: DKINIT
ADDRESS: E821
FUNCTION: DISK SYSTEM INITIALISE

This routine sets flags in the disk RAM buffers that ensure that the directories will be re-read from the disk when next required. It also aborts any open output files but does not update the directories so such files are lost.

Destroys Register A

5.8

NAME: WRITE
ADDRESS: E824
FUNCTION: WRITE CONTIGUOUS LINKED SECTORS FROM RAM

This routine writes a block of RAM directly onto the disk in Zelda format (i.e. with links on the end of each sector) - see Appendix 2.

On input A contains the number of sectors to be written
DE points to the source RAM start address
HL contains the start sector number (in the range 0 to 2001)
(IX-9) holds the ASCII disk drive number ("0" or "1")

On output A contains 0
DE = DE +126*(number of sectors written)
HL contains the last sector number written + 1

Destroys Registers A,D,E,H,L and flags

The disks used on the Zelda system have 77 tracks of 26 sectors each. In the notation used here, the sectors are numbered from 0 (track 0, sector 1) to 2001 (track 76, sector 26), see Appendix 2.

Note this routine only writes 126*A + 2 bytes of RAM onto the disk as link information is added which uses two bytes per sector. The last two bytes of the last sector are written with the data in the RAM and not with a calculated link. If a Zelda format file is being written, these bytes should both contain FF Hex or the file written will not appear to have a legal end.

This routine also does not examine the disk directory and will write over another file if it occupies the sectors called for. The RAM being written is temporarily modified and the routine will leave interrupts enabled.

If this routine is called with A = 1, a full 128 bytes of data will be written to the disk. Thus by repeated calls to this routine, each time with a different value in HL, it is possible to write a file which does not have links on the end of each sector (for example a CP/M (r) disk format). Since DE is set to DE + 126 by this routine, it is necessary to increment DE twice between calls to achieve the desired result.

This routine will abort to PIP if a "DISK FAIL - " error occurs (e.g. NOT READY, TRACK/SECTOR NOT FOUND, etc.).

5.9

NAME: READ
ADDRESS: E827
FUNCTION: READ DISK TO RAM

This routine reads Zelda format linked sectors into RAM. The maximum number of sectors to be read may be specified and the routine will return with an illegal sector address (FFxx Hex) if the end of the file is reached.

On input: A contains the maximum number of sectors to be read
DE points to first position of destination RAM
HL contains start sector number (in range 0 to 2001 - see Appendix 2)

On output: A contains the number of sectors not read (if end of file found)
DE points to next RAM byte to be used
HL contains the next sector address in the file (FFxx Hex if end of file found)

Destroys: Registers A,D,E,H,L and flags

As well as being returned in HL, the link bytes in the last sector read are left in the destination RAM. As a result, n*126+2 bytes of RAM are affected by this routine, where n is the number of sectors read.

This routine can be used to read a disk file that consists of contiguous non-linked sectors. By calling the routine with A = 1, a full 128 bytes of data are read. If this process is repeated, and DE is incremented twice between each call, the destination RAM will contain the data as required.

The routine returns with interrupts enabled and will abort to PIP if a "DISK FAIL - " error occurs (e.g. NOT READY, TRACK/SECTOR NOT FOUND, etc.).

5.10

NAME: BINBCD
ADDRESS: E82D
FUNCTION: BINARY TO BCD CONVERTER

This routine takes a binary number in the HL register pair and returns with a packed BCD number in the range 0 to 9999 in the DE register pair.

On input HL contains the binary number to be converted

On output DE contains the BCD representation of the number

Destroys Registers A,B,C,D,E,H,L and flags

5.11

NAME: STORE
ADDRESS: E833
FUNCTION: STORE DIRECTORY

This routine checks the specified directory in RAM. If valid, it will write it onto the relevant disk and check it can read it again. If necessary, this process will be repeated up to ten times before the routine aborts to PIP with an error message - DIRECTORY ERROR. A DISK FAIL error message may occur.

On input (IX-9) contains the relevant ASCII disk drive number ("0" or "1")

Destroys Registers A,B,C,D,E,H,L and flags

This routine uses WRITE (5.8) and leaves interrupts enabled.

5.12

NAME: GETDIR
ADDRESS: E836
FUNCTION: GET DIRECTORY

This routine checks the directory in RAM and if it finds an error will attempt to read a new directory from the disk in the specified drive.

On input (IX-9) contains the relevant disk drive number ("0" or "1")

On output The disk directory stored in RAM is valid

Destroys Register A and flags

Note that a disk directory is marked invalid by Zelda if the "ready" signal from the associated disk drive disappears. This is how the system detects a disk has been changed.

This routine will abort to PIP if disk error 6,7 or 10 occurs. (See section 3 for an explanation of these numbers).

5.13

NAME: FIND
ADDRESS: E839
FUNCTION: FIND FILENAME IN DIRECTORY

This routine searches the current directory for a specified filename and returns with the sector address found. It must be preceded by a call to the GETDIR routine (5.12) to establish which directory is in use.

On input HL points to the first character of the filename to be found (stored as a 6 character filename + 1 character extension without a delimeter)

On output The zero flag is reset if the file is not found. If the zero flag is set:
HL contains the sector address found (0 - 2001), see Appendix 2
DE-9 points to the directory entry of the file

Destroys Registers A,D,E,H,L and flags

5.14

NAME: FIT
ADDRESS: E83C
FUNCTION: ALLOCATE SPARE AREAS OF DISK

This routine allocates disk storage for a file being written. It scans the directory for the first blank entry and finds the number of sectors available before the next file. The number of sectors allocated will be in the range 1 to 26 as the routine only examines one track at a time. The directory in RAM is updated but not written to the disk.

On input HL points to the first character of the filename (stored as a 6 character filename + 1 character extension without a delimeter)

On output HL contains the sector number stored in the directory entry
DE contains the next sector free to be written
A contains the number of free sectors allocated

Destroys Registers A,D,E,H,L and flags

5.15

NAME: PIPEXT
ADDRESS: E83F
FUNCTION: PERIPHERAL INTERCHANGE PROGRAM EXTERNAL CALL

This routine allows the user to access the commands of PIP (5.4 and Part 2) without control being lost unless an error is found. The string to be interpreted is not typed in by the user but is stored before this routine is called.

On input HL points to command string (stored as it would be typed in in PIP and terminated by a carriage return)

Destroys Registers A,B,C,D,E,H,L,A´,IX and both sets of flags

This routine will abort to PIP if any disk error or command syntax error occurs.

5.16

NAME: PREP
ADDRESS: E842
FUNCTION: READY DISK DRIVE

This routine selects and turns on the relevant disk drive and then waits for up to 2.5 seconds for the "ready" signal to appear. If the drive fails to become ready after this time, the routine will abort to PIP and control will be lost.

On input (IX-9) hold the ASCII disk drive number ("0" or "1")

Destroys Register A and flags

5.17

NAME: RETRY
ADDRESS: E845
FUNCTION: PREPARE FOR ANOTHER ATTEMPT TO READ TRACK/SECTOR HEADER

This routine is closely associated with SEEK (5.18) and effectively provides a way for the user to position the disk drive read/write head using track/sector information rather than sector number.

On input A contains the current error status
B contains the number of tries left + 1 (i.e. 3 means 2 tries)
H contains the track number required (0 - 76)
L contains the sector number required (1 - 26)

On output A = 0 if the drive head was loaded or 4 if not
B = B - 1

Destroys Registers A,B and flags

If the number of tries specified is exceeded, an error message depending on the contents of A will be sent to the console output channel and command will pass to PIP. The message will be :-

DISK n FAIL - (caption)

with the caption depending on which bit(s) of the A register is/are set as follows :-

Bit 7	NOT READY
Bit 6	WRITE PROTECTED
Bit 4	TRACK/SECTOR NOT FOUND
Bit 3	CRC ERROR
Bit 2	LOST DATA

5.18

NAME: SEEK
ADDRESS: E848
FUNCTION: CALCULATE TRACK/SECTOR AND MOVE HEAD

This routine is closely associated with RETRY (5.17) and is used to position the disk drive read/write head using the sector number.

On input HL contains sector number (0 - 2001)

On output A = 0 if the drive head was loaded or 4 if not
B contains (11 - number of tries needed)
H contains track number
L contains sector number

Destroys Registers A,B,H,L and flags

If the sector number called for is out of range, the routine will abort to PIP with an "ILLEGAL TRACK" error message. If the number of retries is exceeded, the message "DISK FAIL - TRACK/SECTOR NOT FOUND" will be displayed and command will again pass to PIP.

5.19

NAME: PBCD
ADDRESS: E84B
FUNCTION: PRINT BCD NUMBER

This routine sends to the console output channel a four digit BCD number with leading zeros blanked (replaced by spaces).

On input HL contains the number to be printed (packed BCD in the range 0 - 9999)

Destroys Registers A,B,D,E and flags

5.20

NAME: DSKACT
ADDRESS: E84E
FUNCTION: SEND COMMAND TO DISK CONTROLLER

This routine sends a command to the FD1771 disk controller and, after a short delay, loops until the command has been executed.

On input A contains the command to be sent to the FD1771

On output A contains the status from the FD1771

Destroys Register A and flags

For a full list of commands available, see the FD1771 data sheet.

5.21

NAME: DELETE
ADDRESS: E851
FUNCTION: DELETE FILE ENTRY

This routine is used to remove an entry in the currently selected directory and will compress the directory if possible by joining together empty entries (see Appendix 2). It must have been preceded by a call to GETDIR (5.12) to determine which disk directory is in use. When a disk file is written, the directory may contain more than one entry associated with the file because of the way the FIT routine (5.14) allocates space. A "continuation" entry is stored with bit 7 of the first character of the filename set. Because this routine only removes one directory entry, it may not remove a file completely from a directory; an example of how to ensure this is given in section 8 of this document.

On input HL-9 points to the directory entry to be removed

On output HL = HL - 9
The directory with the entry removed and compressed if possible

Destroys Registers A,H,L and flags

To remove an entire file from the directory, use the code given in example 8.5 of this handbook.

See entries for ASSIGN (5.5), GETDIR (5.12), FIND (5.13) and FINDL (5.22).

5.22

NAME: FINDL
ADDRESS: E854
FUNCTION: FIND ENTRY IN DIRECTORY

This routine performs the same task as FIND (5.13) except that :-

i) It uses BC instead of HL to point to the filename,
ii) If it cannot find the filename in the directory, it will set bit 7 of the name and try again.

Outputs are as for FIND (5.13) with the addition that bit 7 of the first character of the filename in RAM may be set.

The routine is used to find entries in a directory when deleting and renaming files (see DELETE (5.21)).

5.23

NAME: FINDSR
ADDRESS: E857
FUNCTION: FIND SECTOR ADDRESS

This routine searches the current directory for an entry which has a specified sector number in it. If it finds the entry, it will return pointing to the next directory entry and will reset the carry flag. The routine must be preceded by a call to GETDIR (5.12) to specify the directory and ensure it is valid. NOTE: In version 8.1 of the DOS, FINDSR finds the FIRST entry in the directory containing the specified sector number. In version 8.2 and later versions it finds the LAST entry containing that sector number. Normally there will be only one entry containing a given sector number so this difference is unimportant.

On input DE contains the sector number looked for

On output HL-9 points to the directory entry which has the sector number in it
The carry flag is reset if the sector is found, otherwise reset

Destroys Registers A,B,C,H,L and flags

5.24

NAME: LENGTH
ADDRESS: E85A
FUNCTION: COMPUTE LENGTH OF BLOCK

This routine calculates the length in sectors allocated to a directory entry.

On input HL points to the start of the directory entry

On output DE contains the number of sectors allocated

Destroys   Registers D,E and flags

5.25

NAME: DSKIO
ADDRESS: E85D
FUNCTION: DISK INPUT

This routine performs the same function as DSKIN (5.1) but the user has to set up the parameter block required. This may be useful when adding an extra disk channel for reading. The DSKIN routine is the usual method of reading from a disk file (accessed via RDCHR (4.4)).

With this routine it is only possible to read the first file in a list; DSKIN (5.1) must be used to read from concatenated files.

On input   Bit 3 of E if set will force routine to go back to start of file
IX points to the file's parameter block

Outputs are as for DSKIN (5.1) except that bit 0 of E is not reset.

A description of the format of the parameter block required follows. A user who wishes to set up another disk input channel is strongly advised to create this block by using a call to ASSIGN (5.5) in the normal way, and then to copy the block it produces to another area of RAM. This can, of course, only be done if a spare read channel exists that is to be opened later. Only one filename is allowed when using this routine, rather than a list, so the text string pointers from (IX-14) to (IX-10) are not used. They are listed here for completeness and are used by DSKIN (5.1).

ADDRESS	DESCRIPTION
(IX) to (IX+126*(SECTORS)+1)	DATA BUFFER
(IX-1)	BUFFER POINTER
(IX-2)	FILENAME EXTENSION
(IX-8) to (IX-3)	FILENAME
(IX-9)	DISK DRIVE NUMBER
(IX-11),(IX-10)	TEXT STRING POINTER (current position in filename list)
(IX-13),(IX-12)	TEXT STRING START ADDRESS (start of filename list)
(IX-14)	DEFAULT EXTENSION
(IX-15)	NUMBER OF SECTORS IN BUFFER

In the current Zelda Disk Operating System (Version 8.1), SECTORS = 3 but this value may change later; it is unlikely to be greater than nine in any future system. (The Version number is held at E867 Hex).

This routine will abort to PIP if disk error 1,6,7 or 10 occurs. (See section 3 for an explanation of these numbers). It will also leave interrupts enabled if the disk has to be accessed to refill the buffer.

5.26

NAME: DSKOO
ADDRESS: E860
FUNCTION: DISK OUTPUT

This routine performs the same function as DSKOUT (5.2) but the user has to set up the parameter block required. This may be useful when using an extra disk channel for writing. The DSKOUT routine is the normal method of writing to a disk file (accessed via WRCHR (4.5)).

On input D contains the character to be written
Bit 3 of E if set will open the file
IX points to the file's parameter block

Outputs are as per DSKOUT (5.2) except that bit 0 of E is not set.

The parameter block for an output file is the same as that for an input file (see DSKIO (5.25)) EXCEPT for the following :-

ADDRESS	DESCRIPTION
(IX-11),(IX-10)	FILE OPEN FLAG (set to 01,FE if file is open)
(IX-13),(IX-12)	SECTOR ADDRESS IN DIRECTORY OF LAST BLOCK WRITTEN
(IX-14)	NUMBER OF CONSECUTIVE SECTORS LEFT

These parameters are not usually under user control and are set by a call to FIT (5.14) from within DSKOO whenever the RAM buffer is written to the disk.

No direct call has been provided to close a file that does not use one of the existing disk output channels. However, a call to an address five bytes from the start of the CLOSE routine will enable the user to do this as follows :-

```
LD   HL,(CLOSE+1) ;FIND WHERE CLOSE IS
LD   DE,5
ADD  HL,DE
JP   (HL)
```

with IX addressing the file's parameter block.

This routine will abort to PIP if disk error 2,3,4,5,6,7,9 or 10 occurs. (See section 3 for an explanation of these numbers). It will also leave interrupts enabled if the buffer fills and the disk is written to.

5.27

NAME: SETIX
ADDRESS: E863
FUNCTION: SET IX FOR CHANNEL

This routine loads IX according to channel information contained in the E register. It enables the user to find the parameter blocks set up by ASSIGN (5.5) which may be useful when setting up an extra disk I/O channel (see entries for DSKIO (5.25) and DSKOO (5.26)).

On input Bits 0,1 of E determine channel in use

On output IX points to the RAM area that the Disk Operating System allocates to that channel.

Destroys Register IX and flags

6. OTHER CALLABLE ROUTINES AND DRIVERS

6.1

NAME: STRING
ADDRESS: D95C
FUNCTION: INPUT TEXT LINE

This routine waits for the user to type in a line of text terminated by a carriage return. The characters entered are stored in a RAM buffer whose start address is chosen by the user.

On input HL points to start of RAM line buffer

On output HL points to first character entered (i.e. is unaltered)

Destroys Registers A,B,C,D,E and flags

No count is made of the number of characters entered and the routine will continue to store them until <RETURN> is pressed. The following codes have special meanings :-

<ESC> Pass control to the monitor program and do not return to the calling program

<BACKSPACE> Delete the previous character on the screen and re-use the location where it is stored in the RAM buffer

<RETURN> Store 0D Hex in the RAM buffer and return to the calling program

<DELETE> Print a "\" on the screen and re-use the previous location in the RAM buffer. The "\" is not stored

<CTRL/U> Return to the calling program immediately with a carriage return stored in the first RAM buffer location (i.e. a zero length line)

<TAB> Print between 1 and 8 spaces until the cursor is positioned at one of the preset tab positions (8,16,24,32, etc.). Only the TAB character is stored in the buffer.

All other codes are echoed to the screen and stored.

6.2

NAME: LPTXT
ADDRESS: DDEE
FUNCTION: PRINT TEXT ON VDU

A call to this routine is equivalent to :-

```
LD   E,1
CALL PTXT
```

See entry for PTXT (4.2) for more details.

6.3

NAME: INA
ADDRESS: DEE1
FUNCTION: RAM DRIVER FOR EDITOR

This driver can be used to read from the RAM Editor buffer with line numbers stripped off. The buffer starts at 0000 and is terminated by an ETX after the last record stored. Each record is stored with a carriage return terminator but no line feed. This driver may be used by adding its address as a user mnemonic (see Appendix 4 of part 2) and by copying from it in PIP (see section 5 of part 2). This ensures the pointers are correctly set as PIP will set the initialize flag (bit 3 of E register) when the routine is first called.

On input — Bit 3 of E if set will force the driver to start from the beginning of the buffer.

On output — A and D contain the character read Bits 3 and 7 of E are reset

Destroys — Registers A,D and E

6.4

NAME: INB
ADDRESS: DF05
FUNCTION: RAM DRIVER (READ)

This driver and its companion OUTB (6.5) may be used to read and write data to and from a specific area of RAM as specified by the contents of ENDC (see section 7).

The routine reads a character from the RAM area and increments a pointer to the next location from which to read data. If bit 3 of the E register is set (initialize), the routine will read the first location in the buffer.

On input — Bit 3 of E if set will force the driver to start from the beginning of the buffer

On output — A and D contain the character read
Bits 3 and 7 of E are reset
The pointer is incremented

Destroys Registers A,D,E and flags

The RAM buffer used is from (ENDC) upwards (see section 7). The ENDC pointer is also used by the linking loader and assembler and must be set to a sensible value whenever this driver is no longer used - see the warning notice in section 8.4 of part 2 of the handbook.

6.5

NAME: OUTB
ADDRESS: DF1A
FUNCTION: RAM DRIVER (WRITE)

This driver and its companion INB (6.4) may be used to write and read data from and to a specific area of RAM specified by the contents of ENDC (see section 7).

The routine outputs a character to the RAM area and increments a pointer to the next location to which data will be written. If bit 3 of the E register is set (initialize), the routine will write to the first location in the buffer.

On input D contains the character to be written
Bit 3 of E if set will cause the routine to write to the first location in the buffer.

On output Data is stored in RAM and the pointer is incremented
Bits 3 and 7 of E are reset.

Destroys Registers A,D,E and flags

The RAM buffer used is from (ENDC) upwards (see section 7). The ENDC pointer is also used by the linking loader and assembler and must be set to a sensible value whenever this driver is no longer used - see the warning notice in section 8.4 of part 2 of the handbook.

7. SYSTEM RAM MAP

Reference is made in this section to a "language" when types of program are being distinguished. See Appendix 1 for a description of what constitutes a "language". See also the notes at the end of this section to explain "HIMEM".

ADDRESS	LABEL	
0000 - HIMEM-1		"Language" and "language" workspace
HIMEM- 7FFF		Operating System workspace - do not use
8000 - 9FFF		No ROM or RAM
A000 - A7FF		VDU RAM
A800 - BFFF		No ROM or RAM
C000 - FBFF		Operating system PROM
FC00 - FCFF		Device driver routines. The following drivers supplied on the system disk reside in this area :- RXnnnn, TXnnnn, TOnnnn, PRINT, CPRINT, DPRINT.
FD00 - FDFF		User drivers or monitor extensions. This area should not be used as "language" workspace.
FE00 - FEFF		System workspace (used for example by MON, SUBMIT and SPOOL) - do not use
FF00 - FF03		Editor and Assembler workspace - do not use
FF04 - FF05	ENDC	Buffer start for INB (6.4) and OUTB (6.5)
FF06 - FF07	MEMFLG	Auto mapping indicator - see App. 6 of part 2
FF08 - FF13		Monitor storage - do not use
FF14 - FF15	OPR1	1st operand value produced by SCAN (4.3)
FF16 - FF17	OPR2	2nd operand value produced by SCAN (4.3)
FF18 - FF19	OPR3	3rd operand value produced by SCAN (4.3)
FF1A	OPFLG	Number of operands returned by SCAN (4.3)
FF1B	NXTCHR	Terminator entered into SCAN (4.3)
FF1C	CMD	Last monitor command entered - see ex. 8.4
FF1D - FF1E		Monitor storage - do not use
FF1F - FF20	RTMP	Monitor vector point - see ex. 8.4
FF21 - FF22	HOME	"Home" address of cursor - see ex. 8.3
FF23		VDU ESCape and SI/SO status
FF24 - FF25	ENDMEM	Last contiguous RAM address
FF26		VP: storage - do not use
FF27 - FF28	:CI	Address of console input driver
FF29 - FF2A	:CO	Address of console output driver
FF2B - FF2C	:OI	Address of object input driver
FF2D - FF2E	:OO	Address of object output driver
FF2F - FF30	:SI	Address of source input driver
FF31 - FF32	:SO	Address of source output driver
FF33 - FF8F		Mnemonics and stack - see App. 4 of part 2
FF90 - FF91	POUTB	Pointer for OUTB RAM driver (6.5)
FF92 - FF93	PINB	Pointer for INB RAM driver (6.4)
FF94 - FF95	PINA	Pointer for INA RAM driver (6.3)
FF96 - FFB0		"Language" workspace - Appendix 1
FFB1		STRING workspace - do not use
FFB2 - FFE5		Monitor stack - do not use
FFE6 - FFFF		User's register storage - modified by monitor

In the Disk Operating System PROM, the location E87C (Hex) contains the value of HIMEM. In Version 8.1, for a 32K Zelda, this is 6C53 (Hex).

This PROM also contains two other stored values which address useful variables. These are :-

1) INDEX — This value is stored at E878 (Hex) and is the address of the RAM location at which is stored the address of the first byte of the current disk directory in RAM. In Operating System Version 8.1, The INDEX location contains 79CA (Hex).

2) SELECT — This value is stored at E87A (Hex) and is the address of the RAM location which holds a mimic of the value sent to the disk "power-up" port. The bits have the following meanings :-

Bit 0	If 0, drive 0 is selected; if 1, drive 1 is selected.
Bit 1	If 1, the mains and 24v to drive 0 are switched on.
Bit 2	If 1, the mains and 24v to drive 1 are switched on.

8. EXAMPLES OF USE OF ROUTINES

8.1

```
;
;This example demonstrates the use of ASSIGN
;and GETDF to set up input and output drivers
;
;This program removes any character with bit 7
;set from a disk file.
;
ETX      EQU  3              ;END OF TEXT
CR       EQU  0DH            ;CARRIAGE RETURN
CRLF     EQU  0A0DH          ;CR, LF
;
RENTRY   EQU  0E11DH
PTXT     EQU  0E3C7H
RDCHR    EQU  0E522H
WRCHR    EQU  0E527H
CLOSE    EQU  0E806H
ASSIGN   EQU  0E80CH
GETDF    EQU  0E80FH
GETDIR   EQU  0E836H
FIND     EQU  0E839H
;
STACK    EQU  0FF90H
;
REMOV7:  LD   SP,STACK       ;INITIALISE STACK POINTER
         LD   HL,SOURCE      ;USER PROMPT
         LD   E,2            ;INPUT CHANNEL NUMBER
         LD   D,'S'          ;DEFAULT EXTENSION
         CALL GETDF          ;ASSIGN INPUT CHANNEL
         JP   NC,RENTRY      ;EXIT IF NO INPUT
         CALL FILE           ;TEST FOR FILE
         PUSH HL             ;SAVE FOR LATER
         LD   HL,NTXST       ;ERROR MESSAGE
         JR   NZ,ERROR       ;ABANDON IF NOT THERE
         LD   HL,DESTN       ;USER PROMPT
         LD   E,3            ;OUTPUT CHANNEL NUMBER
         LD   D,'S'          ;DEFAULT EXTENSION
         CALL GETDF          ;ASSIGN OUTPUT CHANNEL
         POP  HL
         LD   DE,BUFFER      ;TEMPORARY STORE
         LD   BC,6
         LDIR                ;COPY FILENAME
         EX   DE,HL
         LD   (HL),CR        ;TERMINATE WITH CR
         LD   HL,BUFFER
         LD   D,'N'          ;DEFAULT EXTENSION
         LD   E,3            ;USED IF NO OUTPUT
         CALL NC,ASSIGN      ;OUTPUT FILE=INPUT FILE.N
         CALL FILE
         PUSH HL
         LD   HL,ALRXST      ;ERROR MESSAGE
         JR   Z,ERROR        ;FILE ALREADY EXISTS
         LD   E,3+1000B      ;INITIAL OUTPUT CHANNEL
         EXX
         LD   E,2+1000B      ;INITIAL INPUT CHANNEL
LOOP:    CALL RDCHR          ;READ BYTE
```

```
        JR   C,EOFILE      ;END OF FILE
        BIT  7,D
        JR   NZ,LOOP       ;SKIP IF BIT 7 SET
        EXX
        LD   D,A
        CALL WRCHR         ;WRITE BYTE
        EXX
        JR   LOOP
EOFILE: EXX
        CALL CLOSE         ;CLOSE FILE JUST WRITTEN
        JP   RENTRY        ;RETURN TO MONITOR
;
ERROR:  LD   E,1           ;CONSOLE OUTPUT CHANNEL
        CALL PTXT          ;PRINT ERROR MESSAGE
        POP  HL            ;FILENAME POINTER
        LD   B,6           ;CHARACTER COUNT
WRLOOP: LD   D,(HL)
        CALL WRCHR         ;PRINT FILENAME
        INC  HL
        DJNZ WRLOOP
        LD   D,'.'
        CALL WRCHR         ;PRINT "."
        LD   D,(HL)
        CALL WRCHR         ;PRINT EXTENSION
        JP   RENTRY
;
;
;FILE - This routine points HL at the compacted filename
;and loads the relevant directory into RAM.  The Z flag
;is set if the file looked for exists.
;   Inputs: IX and Z-flag as output from ASSIGN/GETDF
;  Outputs: HL addresses compacted filename
;           Z-flag set if file exists on disk
; Destroys: A,D,E,H,L,F
;
FILE:   PUSH AF
        CALL Z,GETDIR      ;Z FLAG SET IF DISK
        POP  AF
        PUSH IX
        POP  HL
        LD   DE,-8
        ADD  HL,DE         ;POINT TO FILENAME
        PUSH HL
        CALL Z,FIND        ;IF DISK FILE, CHECK TO
        POP  HL            ;SEE IF IT ALREADY EXISTS
        RET
;
SOURCE: DEFW CRLF
        DEFM 'This program removes all characters with'
        DEFM ' bit 7 set.'
        DEFW CRLF
        DEFM 'Source from ?  '
        DEFB ETX
DESTN:  DEFM 'Destination to ?  '
        DEFB ETX
NTXST:  DEFM 'I cannot find '
        DEFB ETX
ALRXST: DEFM 'I can already find '
```

```
        DEFB ETX
;
BUFFER: DEFS 7
;
        END  REMOV7
```

8.2

```
;This example shows the use of some of the monitor
;routines described in section 4.
;
LPTXT   EQU  0DDEEH
RENTRY  EQU  0E11DH
SCAN    EQU  0E414H
PADD0   EQU  0E604H
;
ETX     EQU  3
CR      EQU  0DH
STOP    EQU  2EH
CARAT   EQU  5EH
CRLF    EQU  0A0DH
;
OPR1    EQU  0FF14H
OPR2    EQU  OPR1+2
OPR3    EQU  OPR2+2
OPFLG   EQU  OPR3+2
NXTCHR  EQU  OPFLG+1
STACK   EQU  0FF90H
;
MONDEM: LD   SP,STACK
GETOPR: LD   HL,PROMPT      ;PROMPT USER FOR INPUT
GETOP1: CALL LPTXT          ;PRINT PROMPT
        CALL SCAN           ;ACCEPT INPUT
        LD   A,(NXTCHR)     ;CHECK TERMINATOR
        CP   STOP           ;ABORT ON '.'
        JP   Z,RENTRY
        CP   CR
        JR   Z,TERMOK
        CP   CARAT
        LD   HL,ILTERM
        JR   NZ,GETOP1      ;TRY AGAIN IF NO GOOD
TERMOK: LD   A,(OPFLG)
        OR   A              ;TEST FOR ZERO
        LD   HL,SOME
        JR   Z,GETOP1       ;REQUEST SOMETHING
        LD   HL,ENTERD
        PUSH AF
        CALL LPTXT
        LD   HL,(OPR1)      ;PRINT ANY VALUES FOUND
        CALL PADD0          ;FIRST VALUE
        POP  AF
        DEC  A
        JP   Z,RENTRY       ;EXIT IF ONE ONLY
        PUSH AF
        LD   HL,(OPR2)
        CALL PADD0          ;SECOND VALUE
        POP  AF
        DEC  A
        JP   Z,RENTRY       ;EXIT IF TWO ONLY
```

```
         LD   HL,(OPR3)
         CALL PADDO         ;THIRD VALUE
         JP   RENTRY        ;EXIT IF THREE
;
PROMPT:  DEFW CRLF
         DEFM 'Please enter up to three operands :'
         DEFB ETX
SOME:    DEFW CRLF
         DEFM 'Please enter something :'
         DEFB ETX
ILTERM:  DEFW CRLF
         DEFM 'Illegal character - please try again :'
         DEFB ETX
ENTERD:  DEFW CRLF
         DEFM 'You entered - '
         DEFB ETX
;
         END  MONDEM
```

In this example, note the following :-

1) The program loads the stack pointer to ensure that a valid RAM area is used for the stack. This means that this routine cannot be used as a callable subroutine.

2) The user is prompted with a text string rather than a simple prompt character. This makes it easier for the operator to understand what is required. The values typed by the user are then stored.

3) A check is made to see if the user has typed in any illegal characters which could cause a wrong value to be returned. This might be very important if a copying command were used, as a wrong value could wipe out the memory !

4) All of the callable routines are referenced by labels, not by addresses. This helps understanding of the program and makes it easier to alter if the address of any of the called routines changes.

8.3

```
;This example shows how to reset the
;HOME screen address to A000 Hex so the
;VDU screen may be used in a memory mapped
;mode.  (See section 8.3 of part 1).
;
FF        EQU  0CH           ;HOME CURSOR
CAN       EQU  18H           ;CLEAR SCREEN
;
CRTCS     EQU  84H           ;6845 REG.  SELECT PORT
CRTCD     EQU  CRTCS+1       ;6845 DATA PORT
;
VDURAM    EQU  0A000H
RENTRY    EQU  0E11DH
WRCHR     EQU  0E527H
HOME      EQU  0FF21H
STACK     EQU  0FF90H
;
SETVDU:   LD   SP,STACK
          LD   A,12          ;SET 6845 HOME
          OUT  (CRTCS),A     ;ADDRESS TO 0
          XOR  A
          OUT  (CRTCD),A
          LD   A,13
          OUT  (CRTCS),A
          XOR  A
          OUT  (CRTCD),A
          LD   HL,VDURAM
          LD   (HOME),HL     ;UPDATE RAM COPY
          LD   E,1001B       ;SEND FF + CAN
          LD   D,FF          ;TO CLEAR SCREEN
          CALL WRCHR
          LD   D,CAN
          CALL WRCHR
          JP   RENTRY
;
          END  SETVDU
```

8.4

```
;This example shows how to add a new monitor
;command.  The "I" command works as follows :-
;
;I <RET>                 Loads a named file to RAM
;                        from 0000 onwards
;
;I NNNN <RET>            Loads a named file to RAM
;                        from NNNN onwards
;
;I NNNN,SSSS <RET>       Loads linked sectors to RAM
;                        from NNNN onwards and starting
;                        from sector SSSS.  Reading
;                        continues until EOF or 256
;                        sectors have been read
;
;I NNNN,SSSS,MM <RET>    Same as above except MM gives
;                        the maximum number of sectors
;                        that will be read (modulo 256)
;
ETX     EQU   3
CRLF    EQU   0A0DH
;
LPTXT   EQU   0DDEEH
RENTRY  EQU   0E11DH
GETDF   EQU   0E80FH
READ    EQU   0E827H
GETDIR  EQU   0E836H
FIND    EQU   0E839H
;
OPR1    EQU   0FF14H
OPR2    EQU   0FF16H
OPR3    EQU   0FF18H
OPFLG   EQU   0FF1AH
CMD     EQU   0FF1CH
RTMP    EQU   0FF1FH
;
        ORG   0FD00H
;
ICMD:   LD    HL,(RTMP)     ;FIND PRESENT JUMP
        LD    DE,START
        XOR   A
        SBC   HL,DE         ;ALREADY EXECUTED ?
        JP    Z,RENTRY      ;QUIT IF SO
        ADD   HL,DE
        LD    (JUMP),HL     ;PASS TO EXISTING ADDRESS
        LD    (RTMP),DE     ;WHEN DONE.  ADD NEW ADDRESS
        JP    RENTRY
;
START:  LD    A,(CMD)
        CP    'I'           ;CORRECT COMMAND ?
        JR    Z,ICMD1       ;CONTINUE IF SO
        LD    HL,(JUMP)
        JP    (HL)          ;ABANDON IF NOT
;
ICMD1:  LD    IX,DEFDK+9    ;SET DEFAULT DRIVE
        LD    A,(OPFLG)     ;HOW MANY OPERANDS ?
        CP    2
        JR    NC,NONAME
```

```
        LD   E,10         ;:OI CHANNEL
        LD   D,'S'        ;DEFAULT EXTENSION
        LD   HL,RDFROM
        CALL GETDF        ;ACCEPT FILENAME
        LD   HL,DKONLY
        JR   NZ,PTEND
        CALL GETDIR
        PUSH IX           ;POINT HL AT FILENAME
        POP  HL
        LD   DE,-8
        ADD  HL,DE
        CALL FIND         ;DOES FILE EXIST ?
        LD   (OPR2),HL    ;STORE START SECTOR IF SO
        LD   HL,NTFND
        JR   NZ,PTEND
NONAME: LD   HL,(OPR2)    ;START SECTOR
        LD   DE,(OPR1)    ;DESTINATION ADDRESS
        LD   A,(OPR3)     ;NUMBER OF SECTORS
        CALL READ
        LD   HL,DONE
PTEND:  CALL LPTXT
        JP   RENTRY
;
RDFROM: DEFM 'Read from: '
        DEFB ETX
DKONLY: DEFM 'Disk files only.'
        DEFW CRLF
        DEFB ETX
NTFND:  DEFM 'File not found.'
        DEFW CRLF
        DEFB ETX
DONE:   DEFM 'Done.'
        DEFW CRLF
        DEFB ETX
;
DEFDK:  DEFB '0'          ;DEFAULT DISK DRIVE
JUMP:   DEFS 2            ;MODIFIED JUMP
;
        END ICMD          ;SPECIFY EXECUTE ADDRESS
;
```

In this example, note the following :-

1) The first part of this program is executed once and installs the monitor extension so that the "I" command will subsequently be recognised. It modifies the contents of the vector stored at RTMP. This vector is the address to which the monitor jumps prior to processing its inbuilt commands. If the monitor extension does not recognize the command letter, it must jump to the address previously stored in RTMP. The program stores this previous vector in RAM at the JUMP location (at the end of the program). To avoid the routine destroying the contents of JUMP if it is executed twice, a check is made to see if RTMP already contains the address of START (the beginning of the new command interpreter) and will not alter the contents of

JUMP again if it does.

2) Each time a monitor command is issued, the program examines the command letter and will jump back to the rest of the monitor processing code via the JUMP vector if the new command is not recognised.

3) If the new command is recognised the number of operands entered (see example 8.2) is checked to see which of the options listed at the beginning of the example is to be used. If a disk file name is to be given, the routine first checks that the file exists before trying to read it.

4) Once the new command has been executed, the routine jumps directly to the monitor restart point. This is done so that a command letter which has already been used can be redefined and the new definition will have priority.

5) For more details of the routines called by this program, see the relevant entries in sections 4 and 5 of this document. Section 7 gives a brief description of the variable stores OPR1, OPR2 etc.

8.5

```
;This example gives the code required to
;delete a file completely from a disk
;directory.  (See DELETE (5.21)).
;
         LD   HL,FILNAM    ;ADDRESS OF FILENAME IN RAM
         CALL ASSIGN       ;SET UP PARAMETER BLOCK
         CALL GETDIR       ;LOAD/CHECK DIRECTORY
         PUSH IX
         POP  HL
         LD   BC,-8
         ADD  HL,BC        ;POINT HL AT FILENAME
         PUSH HL
         CALL FIND         ;CHECK FILE EXISTS
         POP  BC           ;FINDL USES FILENAME AT (BC)
         JP   NZ,NOTFND    ;WARN OPERATOR FILE NOT THERE
DELOOP:  EX   DE,HL
         CALL DELETE       ;REMOVE THIS ENTRY
         CALL FINDL        ;ANY CONTINUATION ENTRIES ?
         JR   Z,DELOOP     ;REMOVE THEM AS WELL
         CALL STORE        ;WRITE DIRECTORY TO DISK
           etc.
```

INDEX

EXAMPLES

APPENDIX 1

Definition of "language"

The term "language" is used in this handbook to denote a particular class of program, with the object of defining which types of program may use which areas of the system RAM. In this context the word does not mean "programming language", which is an abstract concept. "Language" here refers to a particular class of program which is responsible for certain types of tasks. The examples below will hopefully illustrate the point.

There are three main classes of program which run in Zelda. These are:

1. The Operating System.
   This includes all the software which allows other programs to make use of the system hardware via well defined software interfaces (detailed in this handbook), e.g. I/O drivers, file management (DOS), channelised I/O management. As well as the firmware (EPROM) resident routines, user-supplied I/O drivers loaded into RAM are in this category and must not use (or reside in) areas of RAM reserved for the "language".

2. The "Language".
   The "language" is, most simply, the program which makes the system perform a complete task. Typical features of the "language" are:

   a) It is not a subroutine, and cannot be called.
   b) It exits to the monitor command level on completion of the task (or doesn´t exit at all).
   c) It may alter the processor´s stack pointer irreversibly.
   d) It provides the system with its "personality" for the duration of the task.

   There may only be one language in operation at any one time. Examples of "languages" are the resident assembler, resident text editor, VIDIT, TXTPRO, REDCOD, BASIC, LOOK etc.

3. The Applications Program.
   Some "languages" are also applications programs; a program which makes the system perform a specific useful task is an applications program and the example "languages" above are all applications programs except for BASIC. A BASIC interpreter is a "language" and may use the RAM workspace allocated to "languages", but in this case the function of the machine as seen by the user is controlled by the user´s BASIC program. Here the BASIC program is the applications program, not the BASIC interpreter. The BASIC program may not use the language workspace directly without first reserving the space required via the interpreter (e.g. by a DIM statement). An example of such a BASIC applications program is PARCH.

APPENDIX 2

Details of disk file storage

1. Introduction

This appendix describes the disk storage system of the EP1M/27 (Zelda). The disk file and directory formats used are based upon, and are compatible with, those devised for the BENNFAX Scenic Services information storage and retrieval system. Other items of operational equipment which use this format are PRESFAX, HF Transmitter Control Systems, EAGLE (Electronic Announcements, Graphics and Logo Equipment), Monitoring and Information Centre VDUs and EWE (Effects Workshop Equipment).

2. Disk Format

The diskettes used are 8 inch, single-sided, single-density, with 77 tracks, 26 sectors-per-track and 128 bytes-per-sector (IBM 3740 format). The tracks are numbered 0 to 76 (0 being the outermost) and the sectors are numbered 1 to 26, giving 2002 sectors in all or about 250 Kbytes. From the point of view of the Disk Operating System these sectors are considered to be contiguous and are numbered 0 to 2001, corresponding to track 0 sector 1 and track 76 sector 26 respectively. Double-sided disks are used in the MIC VDU, and have been experimentally implemented on a Zelda (with suitable drives), both sides being formatted as above. In this case the logical sector numbering is from 0 to 4003, sectors 0 to 2001 being on side 0 (as above) and sectors 2002 to 4003 on side 1, but with the tracks numbered in reverse order. That is, logical sector 2002 is side 1, track 76, sector 1 and logical sector 4003 is side 1, track 0, sector 26.

3. Disk usage

Track zero is not used at all by the Zelda DOS, i.e. logical sectors 0 to 25 (and 3978 to 4003 in the case of a double-sided disk), although conventionally track 0 sector 1 (logical sector 0) contains details of the ownership of the disk in case it is "lost and found". The remainder of the disk, i.e. tracks 1 to 76, is used for file and directory storage as detailed below. The disk directory begins at logical sector 26 and may be up to 26 sectors (one track) long, according to application (Zelda reserves 13 sectors for the directory); the rest of the disk is available for file storage. The DOS allows for unusable tracks/sectors to be flagged as such in the disk directory so that they will not be written to, although this feature has never been fully utilised (see 6 below).

4. File structure

All files consist of one or more disk sectors, organised as a linked-list. The first 126 bytes of each sector contain data, and the last two bytes contain a link to the next sector in the file, or a special value to indicate that this is the last sector in the file. The link bytes contain the logical sector number of the next sector in the file, least significant byte first, and will be in the range 0 to 7D1 (hex) - 0 to F89 (hex) for a double-sided disk - although values 0 to 26 (hex) are not normally permitted. If the sector is the last in the file, the last byte (i.e. most significant byte of the "link") will have the value FF (hex). The penultimate byte may in future be used to indicate how many bytes of data there are in the last sector, in the range 0 to 126, but in all present systems is set to FF. Note that when a file is read, only the first sector of the file need be located by searching the disk directory, as the rest of the file is found by following the links. Note also that the sectors comprising the file may be scattered on the disk in an arbitrary fashion and may not even be in ascending logical sector order (even if written on a Zelda). The disk directory is a special type of file which consists of a linked-list of sectors, as usual, but which always resides in contiguous sectors beginning at logical sector 26.

5. Disk directory

The disk directory performs two main functions. It contains a list of filenames and associated sector numbers, so that the location of the first sector of a file can be determined, and it acts as a "map" of the disk so that areas of free space can be identified. Although only the first sector of a file needs to be found for that file to be read (subsequent sectors are located by means of the linked-list) the directory must contain information on where every part of the file is located so that, when the file is deleted, the space thereby released can be returned to free space. To this end, the directory consists of a list of entries, each of which corresponds to a block of contiguous sectors on the disk in one of the following categories:

a) A contiguous block of sectors which are not currently allocated to a file and are therefore available for storage of new files ("free space").

b) A contiguous block of sectors which must never be used (such as a track which has failed to format correctly).

c) A contiguous block of sectors belonging to a named file, the first sector of which is the start sector of the file.

d) A contiguous block of sectors belonging to a named file, the first sector of which is NOT the start sector of the file.

e) The end marker of the directory.

These directory entries each consist of nine bytes, the first seven of which indicate the type of entry (and the relevant filename, if appropriate). The last two bytes contain the logical sector number of the first sector of the block (least significant byte first). The entries are in the order of ascending logical sector number, and therefore the directory allows the use of every sector on the disk to be identified. The coding of the five types of entry is as follows:

a) The first seven bytes of the directory entry contain the value zero (00). Only the first byte may be considered significant, although existing Zeldas check the first six bytes. The eighth and ninth bytes hold the logical sector number of the first sector in the block of free space.

b) The first byte contains the value FF (hex), the next six bytes are reserved to indicate the status of the block but are currently unused. The eighth and ninth bytes hold the logical sector number of the first sector in the block of unusable sectors.

c) The first seven bytes of the entry contain the name of the file, in some form. The precise format of the name is system dependent but in the case of Zelda it consists of the six ASCII characters of the filename (padded with spaces if necessary) followed by the single ASCII extension. Bit seven of the first byte of the filename must be reset; the high bits of the other six bytes are reserved for future use as file attribute indicators. The eighth and ninth bytes hold the logical sector number of the first sector in the file.

d) The format of the directory entry is as for (c) above, except that bit seven of the first byte is set. This indicates that the block belongs to part of a file which has been stored as two or more blocks of contiguous sectors, and that this is not the first block.

e) The first byte of the entry contains the value 80 (hex), the next six bytes are reserved to contain a disk name but are currently unused. The eighth and ninth bytes contain the total number of sectors on the disk, i.e. in the case of a single-sided disk they contain 7D2 (hex) - 2002 decimal.

The number of sectors available for file storage, assuming no areas of the disk are marked as unusable, may be found by subtracting the sector number in the first directory entry from the sector number in the last directory entry. In the case of a normal Zelda this is 7D2 (hex) minus 27 (hex) giving 1963 sectors. The length of each block of contiguous sectors may be found by subtracting the sector number in the directory entry corresponding to this block from the sector number in the next directory entry (of whatever type). Therefore, the amount of free space available on the disk may be found by adding together the lengths of all blocks whose directory entry starts with 00; this is how the "Unused" value in PIP´s LIST command is derived.

An example disk directory is shown below:

```
57 4F 4D 42 41 54 53 27 00   first block of file WOMBAT.S
00 00 00 00 00 00 00 51 00   empty space, 10 sectors long
D7 4F 4D 42 41 54 53 5B 00   part of file WOMBAT.S
FF FF FF FF FF FF FF 68 00   unusable block, 26 sectors
D7 4F 4D 42 41 54 53 82 00   part of file WOMBAT.S
00 00 00 00 00 00 00 85 00   empty space, 100 sectors
46 52 4F 47 20 20 53 E9 00   first block of file FROG.S
00 00 00 00 00 00 00 00 01   empty space
80 80 80 80 80 80 80 D2 07   directory end marker
```

It must be emphasised that whilst the directory identifies all sectors on the disk associated with a particular file, it does not include information on the order in which these sectors were written. This information is available only from the links contained within the sectors.

The disk directory is by its nature variable length, being shortest (just two entries) with an empty disk and longest when there are many, or heavily fragmented, files. All systems to date save time by storing only sufficient sectors to contain the current directory, and issue a "directory full" error message if the size of directory exceeds either the RAM space or disk space allocated for it (these may not always be the same - see 7 below). The 13 sectors allocated for the directory on Zelda allow for a maximum of 182 directory entries.

6. Disk initialisation

IBM format disks of the type employed are formatted with all sectors (except those in track 0) containing data E5 (hex). Before use in Zelda or other equipment using a compatible directory structure the disk must be "initialised", i.e. an empty directory is written to logical sector 26. In the case of a standard Zelda the empty directory consists of:

```
00 00 00 00 00 00 00 27 00
80 80 80 80 80 80 80 D2 07
```

i.e. one contiguous block of sectors, 1963 sectors long. As an example of an alternative "empty directory" format, the MIC VDU initialises its double-sided disks as follows:

```
00 00 00 00 00 00 00 34 00
FF FF FF FF FF FF FF D2 07
00 00 00 00 00 00 00 D2 07
80 80 80 80 80 80 80 8A 0F
```

The "dummy" (zero length) block at logical sector 7D2 is present to allow side 0 of the disk to be read on a standard Zelda with version 8.1 DOS. This expects to see a directory entry containing 7D2 otherwise a DIRECTORY ERROR message is produced. Version 8.2 of the DOS, as used in a double-sided Zelda, has had this restriction removed. The sector number 34 in the first entry indicates that a full track (26 sectors) is reserved for the directory.

Note that, strictly speaking, a disk initialising operation should check that all sectors on the disk are accessible (properly formatted and readable) and if not should create appropriate directory entries to prevent subsequent use of the bad sectors. This has never been found necessary in practice.

7. Compatibility between systems

The degree of compatibility between disks written by different systems (e.g. Zelda on the one hand and the MIC VDU on the other) is affected by four factors: file contents, file names, directory size and disk size.

Whether file names and file contents make sense to the system in question will obviously depend on the nature of the system. Because Zelda is a general purpose computer it will usually be possible both to read and to write files in accordance with the requirements of another system. On the other hand, a standard Zelda text file, for example, may make no sense to an HF Control System VDU. File name incompatibility could pose a problem if the system in question uses the high bits of the filename (except the first byte) for its own purposes.

Directory size will cause a compatibility problem if the directory on the disk is bigger than the space allocated for the directory in RAM. This might occur if an attempt was made to read an MIC VDU disk on a Zelda. Generally speaking this is not too much of a problem as the directory size only approaches the limit imposed by the disk when the disk is very full and/or the files are very fragmented.

A double-sided system (e.g. MIC VDU) should have no difficulty reading and writing a single-sided disk. The disk directory includes information on the disk size and prevents the system attempting to access the second side. However, although side 0 of a double-sided disk may be read by a single-sided Zelda (with DOS version 8.1) it should not be written to as the Zelda may fail to write the complete directory back to the disk. As a rule this cannot in any case occur because the different position of the index hole in a double-sided disk prevents it being accepted by a single-sided drive.

APPENDIX 3

Input/output channel usage

As described in section 4.3 of part 2 of this handbook, there are six input/output channels available. They are generally addressed by three bits of the E register as follows:

Channel	Bit 2,E	Bit 1,E	Bit 0,E
Console input	0	0	0
Console output	0	0	1
Object input	0	1	0
Object output	0	1	1
Source input	1	0	0
Source output	1	0	1

In the case of the "object" and "source" channels the names are largely historical and correspond only to the conventional use of the channels by the system firmware (assembler, loader etc.). The console input and console output channels (0 and 1) are reserved for this purpose and are vectored to special routines which perform various system housekeeping utilities. They should not be reallocated to different drivers without considerable thought, and should always be restored to their default values (see entries for TTID (4.16), VDUOUT (4.17) and Appendix 5 of part 2 of this handbook).

The "object" and "source" channels may be used for general purpose input/output within user programs, but the following points should be noted:

1. Some "background" programs (e.g. SPOOL and SUBMIT) use the source input and output channels for their own purposes. The user is therefore advised to use the object input and output channels in preference, and to use the source channels only when one channel of each type is insufficient.

2. The user should, in general, define the channel in use by setting all of bits 0, 1 and 2 of the E register to the appropriate values, even if the particular driver in use does not need all three:

   Bit 0: The DSKIN (5.1) and DSKOUT (5.2) routines do in fact set bit 0 of E to a defined state (0 for DSKIN, 1 for DSKOUT) as a side effect, but other driver routines should not test or alter it. In general, any driver accessed via RDCHR (4.4) or WRCHR (4.5) should not need to use bit 0 of E but a direct call to a routine might; for example, bit 0 of E will determine whether ASSIGN (5.5) and GETDF (5.6) opens a file for input or output.

   Bit 2: Bit 2 of E is not used directly by DSKIN or DSKOUT, but the normal access to these routines via RDCHR or WRCHR respectively requires that bit 2 be set to determine the channel to use. Any call directly to DSKIN, DSKOUT, DSKIO or DSKOO routines with the E register selecting a "console" channel (0 or 1) will in fact result in the use of the corresponding "source" channel (4 or 5).

3. The system provides no protection against a call to RDCHR or WRCHR with the three least significant bits of E set to an illegal value (6 or 7) and a system crash is likely (see section 7 and appendix 4 of part 2).