DESIGNS DEPARTMENT

DESIGNS DEPARTMENT HANDBOOK

No. 2.466(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 3 - Utilities

BRITISH BROADCASTING CORPORATION
ENGINEERING DIVISION

Issue 1
12/12/83

DESIGNS DEPARTMENT HANDBOOK

No. 2.466(82)

ZELDA Development System

EPlM/27

Users' Handbook

Part 3 - Utilities

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

Written by: D. J. King
J. Robinson
R. T. Russell
P. A. Stevens

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.

Issue 1
12/12/83

DESIGNS DEPARTMENT HANDBOOK

No. 2.466(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 3 - Utilities

CONTENTS

DESIGNS DEPARTMENT HANDBOOK

No. 2.466(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 3 - Utilities

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 (this document)

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

Part 4 - Writing Applications Software 2.467(82)

Describes in detail the resident routines available to the user, and 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. Notes on loading and running programs

Utility programs are generally stored on disk in one of two formats, Intel-format object code (extension .O) or C-format binary (extension .C). Details of these two formats can be found in Part 2 of this Handbook. The method of loading and running a program depends on which format it is in:

3.1 Intel format

This format is used preferentially for short programs as it includes additional protection against corruption (checksums) and allows programs to be relocatable. This latter feature is of particular value with utility programs which perform manipulation of data stored in memory (e.g. LOOK, REDCOD) in which case the utility itself must be loaded away from the area of interest. It should be noted that not all programs provided in Intel format are necessarily relocatable; the normal load address should be used unless there is a good reason not to.

Intel format programs are loaded using the monitor L command. As a general rule the linking loader, with an offset of zero, should be used as this avoids the necessity to know the execute address of the program:

```
.L 0cr
LOAD FROM? dev:filnam.e cr
BEG ADDR ????
EXECUTE  ????
END ADDR ????
UNDEF SYM 00
#Ecr
```

You should be aware that the use of the linking loader will result in the alteration of memory contents other than those occupied by the program being loaded. The loader's symbol table is written, in the case of a 32K ZELDA, from address 1FFF (hex) downwards. If it is important that these locations be left unaltered, an explicit address for the loader's symbol table should be entered as the second operand of the L command, e.g.

```
.L 0,5000cr
```

In the above example, the symbol table will be written from 4FFF (hex) downwards. The specified address must not exceed 6C52 (see handbook part 2).

If it is desired to load a program other than at its usual load address (generally 0000) then an offset should be specified in the load command. For example, to load LOOK at 4000 (hex) the following command should be given:

```
.L 4000cr
LOAD FROM? LOOK cr
```

A small number of Intel format utility programs cannot be loaded with the linking loader as they reside at an address outside of the loader's valid range (e.g. above FC00 hex) and an ERROR 4 is produced. In these cases the absolute loader should be used:

```
.L cr
```

In all such cases the program is not intended to be executed by the user so the absence of an execute address is unimportant.

3.2 C-Format

C-format files are used in the case of long programs, where the saving of space and loading time is valuable, and when the program must be loaded and executed outside of the address range of the linking loader. In general, such programs should be executed using the monitor X command rather than loaded using the G command:

```
.X cr
EXECUTE BINARY FILE: dev:filnam.e cr
```

4. CMPARE - File Comparator

File name:	CMPARE.O
Description:	Compares two files and displays parts which differ.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	940 contiguous bytes, including workspace.
Commands:	<RETURN> - Continue comparison.
	A1/A2 - Advance by one line in file 1 / file 2.
	V1/V2 - View current line in file 1 / file 2.
	R1/R2 - "Rewind" file 1 / file 2.
	<ESC> - Exit to system monitor.

4.1 Description

CMPARE performs a line by line comparison between two files (including binary files) and displays any pair of lines in which differences are found. A facility is provided for manual resynchronisation of the files so that two similar files may be compared successfully, even if lines which are present in one are absent from the other. CMPARE is particularly useful for verifying that an edit session has resulted in only the expected changes to a file.

4.2 Preliminaries

Load CMPARE and execute it at the load address. Load the disk(s) containing the files to be compared into the drive(s).

4.3 User Input

Reply to the prompts as follows:

FIRST FILE: Enter the name of one of the files to be compared.
Default drive: DK0:
Default extension: .S

SECOND FILE: Enter the name of the other file to be compared.
If just <RETURN> is entered then:
  Drive: DK0:
  Filename: Same as FIRST FILE
  Extension: .N
otherwise:
  Default drive: DK0:
  Default extension: .S

BINARY FILES? Reply Y or N; <RETURN> is not required. The default is N.
Y should be used for files which may contain the ETX character (ASCII 03) as a valid data byte.
N should be used for files which use the ETX character as an end-of-file marker.

IGNORE LINE FEEDS? Reply Y or N; <RETURN> is not required. The default is N. This option is only given for non-binary operations.
Y should be entered if the line feed character (0A hex) is to be ignored by the comparator. This allows the successful comparison of two nominally identical files where one file has lines terminated by CR (0D hex), and the the other file has lines terminated by CRLF (0D 0A hex).
N should be entered if all characters, including LF, are to be verified.

4.4 User Output

If the two files are identical the message FILES IDENTICAL is displayed and the program returns to the system monitor. If a difference is found the two lines in disagreement are displayed, together with their line numbers, followed by a prompt character ">". CMPARE then awaits a command (see 4.5 below), for example:

```
0002 second line in file one
0002 second line in file 2
>
```

(a) The first line of a pair is from the "FIRST FILE" and the second line from the "SECOND FILE", as declared in the preliminaries.

(b) Control characters are displayed by the symbols shown in Appendix 1. Characters with bit 7 set to 1 are displayed in inverse video. This is the same convention as is used by the driver DISFUN (see section 27) and by VIDIT (section 16).

(c) The line number is incremented when a carriage return (0D hex) is read from a file. This applies in both binary and non-binary modes.

(d) Files are compared line by line, or by blocks of 74 characters if the line terminator (CR) is not encountered earlier. All characters in a file are verified and all discrepancies displayed, even if the lines (separated by CR) are longer than 74 characters. Note that the displayed line number is only incremented when CR is read.

When the ends of both files have been reached, the message REMAINDER OF FILES IDENTICAL is displayed, and the program exits to the system monitor.

4.5 Commands

When a difference is found between the files, the relevant pair of lines is displayed followed by a prompt (">"). The commands given below may then be used.
Note that "line" in this context means line of characters terminated by CR or block of 74 characters, whichever is the shorter.

<RETURN>	Continue comparison from next pair of lines.
A1<RETURN>	Advance the first file by one line and display the line read.
A2<RETURN>	Advance the second file by one line and display the line read.
V1<RETURN>	Display the latest line read from file 1.
V2<RETURN>	Display the latest line read from file 2.
R1<RETURN>	Reset file 1 to start reading from the beginning.
R2<RETURN>	Reset file 2 to start reading from the beginning.
<ESC>	Quit program and return to monitor.

Note that the line displayed as a result of using the Advance command (A1/A2) is not compared with a line from the other file. When this command is used to resynchronise files the user should check that the current lines in the two files are identical by using the View command (V1/V2).

4.6 Example

The example shows the result of comparing the two text files shown below. The symbols for CR and LF have been omitted for clarity.

FILE1	FILE2
Line 1	Line 1
Line 2	Line 2 with modifications
Line 3	Line 3
Line 4	Extra line
Line 5	Line 4
	Line 5

```
FIRST FILE: FILE1cr
SECOND FILE: FILE2cr
BINARY FILES? N
IGNORE LINE FEEDS? N
0002 Line 2
0002 Line 2 with modifications
>cr
0004 Line 4
0004 Extra line
>A2cr
0005 Line 4
```

[this line is the same as the current line in file 1, so the files are now resynchronised and the comparison may continue]

```
>cr
REMAINDER OF FILES IDENTICAL
.
```

5. MFT - Multi File Transfer

File name: MFT.O
Description: Copies multiple disk files.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: All available memory is used for buffering.

5.1 Description

MFT enables the rapid copying of files (including binary files) from a disk to another disk or output device. Its speed is achieved by using all of the available system RAM as a buffer so that, usually, several files can be read and written in a single operation. Disk-to-disk transfer can use either one or two drives.

5.2 Preliminaries

Load and execute MFT (Loading at 0 will allow the largest possible buffer, since MFT uses RAM from its own address upwards to the same limit as the Editor).

Place the disk containing the files to be copied in drive 0 and, if two drives are available (for disk-to-disk copying), the destination disk in drive 1. Note that it is possible to copy from drive 1 to drive 0, but this requires great care when replying to the LIST FILES prompt.

5.3 User Input

Reply to the prompts as follows:

List files:
Enter the list of files to be copied, separated by commas or spaces and terminated by RETURN. The default extension is .S, and the wildcard character ("?") is permitted within filenames; e.g. A?????.L will copy all .L files whose names begin with "A". Typing just DK0: will result in all .S files being copied. Note that the rules governing default filenames are non-standard: FILE.A,.B means FILE.A,??????.B and, if DK1: is the source device, all filenames must be explicit and preceded by DK1:. Devices other than disk drives are not permitted here.

Destination:
Enter DK1: to initiate copying to that drive. Alternatively, enter no parameters to initiate single-drive copying, or a device mnemonic to copy to that device. In the latter case, all files will be copied sequentially, with the initialise bit in Register E set on the first entry to the device driver for each file.

5.4 User Output

Buffer size = nnn sectors
    indicates the amount of RAM which will be used for buffering.

Buffer size = insufficient
    means that there is not enough space to buffer one sector (unlikely!).

<filename> nnn sectors read [written]
    MFT has read [written] this many sectors of the named file to [from] the buffer.

Insert source [destination] disk & press any key
    prompts you to change disk (in single-drive mode).

5.5 Example

The example illustrates copying all files with an extension of .S from disk 0 to disk 1:

```
Multi-File Transfer utility V1.0
List files: .S cr
Destination? DK1: cr
Buffer size = 213 sectors
INITDD.S -  10 sectors read
VDUSET.S -   8 sectors read
JAB   .S -   8 sectors read
MARGIN.S -   1 sectors read
MARG65.S -   1 sectors read
INITDD.S -  10 sectors written
VDUSET.S -   8 sectors written
JAB   .S -   8 sectors written
MARGIN.S -   1 sectors written
MARG65.S -   1 sectors written
.
```

6. MFC - Multi File Compare

File name: MFC.O
Description: Compares multiple disk files.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: All available memory is used for buffering.

6.1 Description

MFC enables the rapid comparison of files (including binary files) between one disk and another. Its speed is achieved by using all the available system RAM as a buffer so that, usually, several files can be read and compared in a single operation. MFC can be used in either a single-drive or dual-drive mode.

6.2 Preliminaries

Load and execute MFC (Loading at 0 will allow the largest possible buffer, since MFC uses RAM from its own address upwards to the same limit as the Editor).

Place one disk containing the files to be compared in drive 0 and, if two drives are available, the other disk in drive 1.

6.3 User input

Reply to the prompts as follows:

List files:
Enter the list of files to be compared, separated by commas or spaces and terminated by RETURN. The default extension is .S, and the wildcard character ("?") is permitted within filenames; e.g. A?????.L will compare all .L files whose names begin with "A". Typing just DK0: will result in all .S files being compared. Note that the rules governing default filenames are non-standard: FILE.A,.B means FILE.A,??????.B and, if DK1: is specified, all filenames must be explicit and preceded by DK1:. Devices other than disk drives are not permitted.

Comparison source:
Enter DK1: or DK0: as appropriate to initiate dual-drive mode, or alternatively enter no parameters to initiate single-drive mode.

6.4 User output

Buffer size = nnn sectors
indicates the amount of RAM which will be used for buffering.

Buffer size = insufficient
means that there is not enough space to buffer one sector (unlikely!).

<filename> nnn sectors read [compared]
MFC has read [compared] this many sectors of the named file to [with] the buffer.

Insert first [second] source disk & press any key
prompts you to change disk (in single-drive mode).

If a difference is found, the message "COMPARISON FAILED" is issued and control is returned to the monitor. The file in which the difference occurred is the last one whose name is displayed.

6.5 Constraints

MFC does not check that files are the same length. If the file on the "comparison" disk is longer than, but otherwise identical to, the file on the "source" disk, the comparison will not fail.

6.6 Example

This example illustrates comparing all .S files on disk 0 with files of the same name on disk 1:

```
Multi-File Compare utility  V1.0
List files: .S cr
Comparison source? DK1: cr
Buffer size = 213 sectors
INITDD.S -  10 sectors read
VDUSET.S -   8 sectors read
JAB   .S -   8 sectors read
MARGIN.S -   1 sectors read
MARG65.S -   1 sectors read
INITDD.S -  10 sectors compared
VDUSET.S -   8 sectors compared
JAB   .S -   8 sectors compared
MARGIN.S -   1 sectors compared
MARG65.S -   1 sectors compared
.
```

7. MFTCPM - Multi File Transfer from CP/M disk

File name: MFTCPM.O
Description: Copies multiple files from a CP/M disk to a ZELDA disk.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: All available memory is used for buffering.

7.1 Description

MFTCPM enables the rapid copying of files (including binary files) from a disk recorded in standard CP/M format (8 inch single density) to a ZELDA disk or output device. Disk-to-disk transfer can use either one or two drives.

7.2 Preliminaries

Load and execute MFTCPM at 0. Insert the CP/M disk into drive 0 and, if two drives are available (for disk-to-disk copying), the destination disk in drive 1. Note that it is possible to copy from drive 1 to drive 0 but this requires great care when replying to the LIST FILES prompt.

7.3 User input

Reply to the prompts as follows:

List files:
Enter the list of files to be copied, separated by commas or spaces and terminated by RETURN. File names must adhere to CP/M conventions (8 character filename plus optional 3 character extension) and may include wildcards ("?" and "*"), e.g. A*.PRN or A???????.PRN will copy all .PRN files whose names begin with "A". If drive 1 is the source device, all filenames must be explicitly preceded by "B:".

Destination:
Enter DK1: or DK0: as appropriate to initiate copying to that drive. Alternatively, enter no parameters to initiate single-drive copying, or a device mnemonic to copy to that device. In the latter case, all files will be copied sequentially, with the initialise bit in register E set on first entry to the device driver for each file.

7.4 User output

Buffer size = nnn sectors
indicates the amount of RAM which will be used for buffering.

Buffer size = insufficient
means that there is not enough space to buffer one sector (unlikely!).

<filename> nnn sectors read [written]
MFTCPM has read [written] this many sectors of the named file to [from] the buffer.

Insert source [destination] disk & press any key
prompts you to change disk (in single-drive mode).

7.5 Constraints

Filenames are truncated to the first six characters and extensions to the first character before being written to the ZELDA disk. If two or more of the specified CP/M files have the same name after truncation in this way then a FILE ALREADY EXISTS error will occur.

Because CP/M uses records of 128 bytes and ZELDA uses 126 bytes, a file copied using MFTCPM may have up to 125 NULs appended to its end. The normal CP/M end-of-file character (ASCII files) is CTRL/Z (1A hex) whereas the equivalent with ZELDA files is CTRL/C (03 hex). MFTCPM takes no account of this difference as it copies all files as binary files.

7.6 Example

This example illustrates copying all .Z80 files from a CP/M disk in drive 0 (drive A in CP/M terms) to a ZELDA disk in drive 1:

```
CP/M to BBC Transfer utility V1.0
List files: *.Z80 cr
Destination? DK1: cr
Buffer size = 192 sectors
TERMINAL.Z80 -  43 sectors read
ZPILOT  .Z80 -  71 sectors read
SETUP   .Z80 -   4 sectors read
TERMINAL.Z80 -  43 sectors written
ZPILOT  .Z80 -  71 sectors written
SETUP   .Z80 -   4 sectors written
.
```

Note that on the destination disk the files will be called TERMIN.Z, ZPILOT.Z and SETUP.Z respectively.

8. BACKUP - Entire disk backup utility

File name: BACKUP.O
Description: Copies all files on a disk onto another disk.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: All available memory is used for buffering.

8.1 Description

BACKUP copies all files from the disk in drive 0 onto the disk in drive 1, and then compares all the copied files with the originals. It is intended to be used to produce backup copies of a disk, but also has the valuable side-effect of compacting the disk directory, and improving access times, because the files on the destination disk are not fragmented. All available system RAM is used as a buffer.

8.2 Preliminaries

Load BACKUP from the utilities disk (loading at 0 will allow the largest possible buffer, since BACKUP uses RAM from its own address upwards to the same limit as the Editor).

Place the disk to be copied in drive 0, the destination disk in drive 1, and execute BACKUP (linking loader E command). The destination disk should not normally contain any files, but if it does these will be unaffected by BACKUP.

8.3 User input

None.

8.4 User output

Buffer size = nnn sectors
indicates the amount of RAM which will be used for buffering.

Buffer size = insufficient
means that there is not enough space to buffer one sector (unlikely!).

<filename> nnn sectors read [written] [verified]
BACKUP has read [written] [verified] this many sectors of the named file to [from] [against] the buffer.

If the destination disk is found to contain a file of the same name as one to be written, the FILE ALREADY EXISTS message is issued and control is transferred to PIP.

8.5 Example

This example illustrates backing up of a disk containing only four files:

```
Entire disk backup utility V1.0
Buffer size = 212 sectors
ZELINI.S -  56 sectors read
ZDOS32.S - 100 sectors read
ZCPU32.S -  43 sectors read
ASMB80.O -  12 sectors read
ZELINI.S -  56 sectors written
ZDOS32.S - 100 sectors written
ZCPU32.S -  43 sectors written
ASMB80.O -  12 sectors written
Buffer size = 212 sectors
ZELINI.S -  56 sectors read
ZDOS32.S - 100 sectors read
ZCPU32.S -  43 sectors read
ASMB80.O -  12 sectors read
ZELINI.S -  56 sectors verified
ZDOS32.S - 100 sectors verified
ZCPU32.S -  43 sectors verified
ASMB80.O -  56 sectors verified
.
```

9. COPYDK - Copy disk

File name: COPYDK.O
Description: Makes an identical copy of a disk.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: 3668 contiguous bytes.

9.1 Description

COPYDK copies every sector of the disk in drive 0 onto the same sector of the disk in drive 1, i.e. it copies 26 sectors on each of 77 tracks. Because it makes no assumptions about file or directory formats, COPYDK can be used to copy disks other than standard Zelda-format disks, e.g. CP/M disks. The only requirement is that the disks should be 8-inch IBM-format, soft-sectored, single-density, 128 bytes per sector. COPYDK does not verify that the data has been written to the destination disk successfully. Although faster than BACKUP, it is not recommended that COPYDK be used to make backup copies of Zelda-format disks.

9.2 Preliminaries

Load COPYDK at address 0. Place the disk to be copied in drive 0, the destination disk in drive 1, and execute COPYDK (E command). The destination disk should normally be an unused (formatted) disk. Any data on the destination disk will be DESTROYED by COPYDK.

9.3 User input

Respond to the CONFIRM COPY DISK (Y FOR YES) prompt with "Y" to carry out the operation. Any other character will abort back to the monitor.

9.4 User output

None.

9.5 Example

This example illustrates copying of a disk:

```
*** CAUTION!  ***
THIS ROUTINE COPIES ALL OF DISK 0 ONTO DISK 1.  THE PREVIOUS
CONTENTS OF DISK 1 ARE DESTROYED.
CONFIRM COPY DISK (Y FOR YES):
.
```

10. SUMCHK - Calculate checksum byte

File name:	SUMCHK.O
Description:	Reads an "Intel" format object code file and computes the value to which a byte of 0FFH should be changed to make the modulus-256 sum equal to zero over a specified address range.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	371 contiguous bytes.
Commands:	<ESC> exit to system monitor (during filename entry)

10.1 Description

SUMCHK reads an Intel format object code file and displays the value to which a byte of 0FFH should be changed in order to make the 8-bit checksum over a specified address range equal to zero. Any bytes within the address range which are not specified by the file are taken to have value 0FFH. The address range may be specified, or by default a value is computed for each 2K byte block of the file.

10.2 Preliminaries

Load SUMCHK and execute it at the load address. Insert the disk containing the Intel format file to be processed into a drive.

10.3 User Input

Reply to the prompts as follows:

OBJECT FILE TO SUM?

Enter the name of the file to be processed.
Default drive: DK0:
Default extension: .O
<ESC> exits to the monitor.

ADDRESS RANGE (START,END)
OR <RETURN> FOR AUTO:

Enter the address range in hexadecimal over which the checksum is to be calculated, followed by <RETURN>. The entry format is the same as for monitor commands, i.e. only the last four digits of each address are significant and the addresses should be separated by a space or comma. The checksum is calculated from the start address to the end address inclusive.

<RETURN> alone will result in the checksum being calculated for each 2K byte block of the file.

10.4 User Output

Once the file has been read the result is displayed in the following format:

To correct sum for address range aaaa to bbbb change FF to cc

where aaaa,bbbb are the start and end addresses and cc is the new value to replace 0FFH to make the 8-bit sum zero.

If "auto" mode has been used and the file is longer than 2K bytes then two or more such lines will be displayed. On completion the program exits to the monitor.

10.5 Example

```
CHECKSUM CALCULATOR
OBJECT FILE TO SUM? SUMCHK cr
ADDRESS RANGE (START,END) OR <RETURN> FOR AUTO: 0,7FF cr
To correct sum for address range 0000 to 07FF change FF to  4D
.
```

11. REDCOD - "Redcode" calculator

File name:	REDCOD.O
Description:	Calculates the Redcode of the contents of an area of memory.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	92 contiguous bytes.
Commands:	. exit to system monitor (during address entry)

11.1 Description

REDCOD calculates the Redcode of a specified area of memory and displays the result. "Redcode" is the name given to the cyclic redundancy code (CRC) used by the BBC in confirming that the contents of a PROM are correct. It is a 16-bit number derived from the data in the memory, and is sensitive to both the value and the order of the data. For a complete description of the Redcode algorithm see Designs Department Technical Memorandum 1.155(80) "BBC Code of Practice for the Use of PROMs".

11.2 Preliminaries

Load REDCOD and then load into memory the data whose Redcode is to be calculated. Note that the data memory must not overlap the program itself. Because the relocating linking loader uses part of the memory for its scratchpad it is always safer to load the data AFTER loading REDCOD, and to use the absolute loader to load the data. If the data whose Redcode is wanted is to be programmed into a PROM, and if some bytes in the PROM will be left unprogrammed, it may be necessary to fill the appropriate memory address range with the data value corresponding to an unprogrammed PROM (usually 0FFH) before loading the data.
Execute the utility at its load address.

11.3 User Input

Reply to the prompts as follows:

REDCODE FOR ADDRESS RANGE:aaaa,bbbb cr

where aaaa and bbbb are the first and last addresses, in hexadecimal, of the block of memory whose Redcode is to be calculated. The entry format is the same as for the monitor commands: i.e. only the last four digits of the address are significant and the values should be separated by a space or comma.
Full stop (.) will exit to the monitor.

11.4 User Output

After a short pause the Redcode is displayed and the program exits to the system monitor.

11.5 Example

```
.L 5000 cr
LOAD FROM: REDCOD cr
BEG ADDR 5000
EXECUTE  5000
END ADDR 505C
UNDEF SYM 00
#.
.F 0,7FF,FF cr
.L cr
LOAD FROM: DATA cr
.E cr
REDCODE FOR ADDRESS RANGE: 0,7FF cr
ABCD
.
```

12. SHRINK - Replace spaces with tabs in source code file

File name: SHRINK.O
Description: Replaces spaces with tab characters in an assembly language source code file.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: 289 contiguous bytes.

12.1 Description

SHRINK can be used to reduce the length of a source code file in which spaces have been used to align the columns of labels, op-codes, operands and comments. SHRINK operates by replacing each sequence of spaces by an HT character (ASCII 09). SHRINK is suitable for use only with assembly language source files as the comment delimiter is assumed to be a semi-colon (;) and the string delimiter a single quote (').
The line editor, assembler and screen editor (VIDIT) are all compatible with files processed by SHRINK and assume tab-stop positions at columns 8, 16, 24, 32 etc. Listing files produced by the assembler do not include tab characters as the assembler expands them to spaces, so it is not necessary for a printer to respond to tab characters in a particular fashion.

12.2 Preliminaries

Load SHRINK and execute it at its load address. Insert the disk containing the source program to be processed into one of the disk drives (normally DK0:). If the processed file is to be written to another disk, insert the destination disk into the other drive.

12.3 User Input

Reply to the prompts as follows:

FROM? Enter the source file name. Default extension is .S
<ESC> will exit to the system monitor.

TO? Enter a name for the "shrunk" file. Default extension is .N
<ESC> will exit to the system monitor.

12.4 User Output

None.

12.5 Example

```
FROM? WOMBAT cr
TO? WOMBAT cr
.
```

The above input to SHRINK will result in the creation of a file WOMBAT.N, containing tab characters, from an input file WOMBAT.S, assumed to contain spaces or a mixture of both spaces and tabs.

13. TABLAT - Source code formatter

File name:	TABLAT.O
Description:	Formats an assembly language source file by inserting suitable numbers of space characters.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	250 contiguous bytes.
Commands:	<ESC> exit to system monitor (during filename entry).

13.1 Description

TABLAT replaces all occurrences of space or tab characters in a file (unless in a comment or within a literal text string) with the correct number of space characters such that opcodes are tabulated at column 8, operands at column 13 and comments at column 25. Trailing space characters are stripped. TABLAT is suitable only for assembly language source files as the comment delimiter is assumed to be a semi-colon (;) and the string delimiter a single quote (´). A file processed by TABLAT will be significantly longer than the same file processed by SHRINK.

13.2 Preliminaries

Load TABLAT and execute it at its load address. Insert the disk containing the source program to be processed into one of the disk drives (normally DK0:). If the processed file is to be written to another disk, insert the destination disk into the other drive.

13.3 User Input

Reply to the prompts as follows:

FROM? Enter the source file name. Default extension is .S. <ESC> will exit to the system monitor.

TO? Enter a name for the tabulated file. Default extension is .N. <ESC> will exit to the system monitor.

13.4 User Output

None.

13.5 Example

```
FROM? WOMBAT cr
TO? WOMBAT cr
.
```

The above input to TABLAT will result in the creation of a tabulated file WOMBAT.N (containing spaces) from an untabulated file WOMBAT.S (containing spaces, tabs or a mixture of both).

14. CLEAN - Remove unwanted characters from ASCII file

File name:	CLEAN.O
Description:	Resets bit 7 of all characters in a file, and removes all control characters other than carriage return, line feed and nul.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	196 contiguous bytes.
Commands:	<ESC> exit to system monitor (during filename entry).

14.1 Description

CLEAN processes an ASCII text file by resetting the most significant bit of all characters and filtering out all control characters other than carriage return, line feed and nul. Its main use is to clean up a file produced by the Text Processor (TXTPRO) so that it is suitable for sending to a printer which uses bit 7 to indicate a graphics character. TXTPRO sets bit 7 of those characters which it has inserted to pad out lines to a constant width (justification), allowing such characters to be distinguished from spaces in the original text. CLEAN may also be used to remove control codes such as those which might have been inserted to control special printer features such as underlining or bold print.

14.2 Preliminaries

Load CLEAN and execute it at its load address. Insert the disk containing the source file to be processed into one of the disk drives (normally DK0:). If the processed file is to be written to another disk, insert the destination disk into the other drive.

14.3 User Input

Reply to the prompts as follows:

FROM: Enter the source file name. Default extension is .S
<ESC> will exit to the system monitor.

TO: Enter a name for the tabulated file. Default extension is .N
<ESC> will exit to the system monitor.

14.4 User Output

None.

14.5 Example

```
This program removes all control characters
except CR, LF and NUL from an ASCII file.
It also resets bit 7 of all characters.
FROM: WOMBAT cr
TO: WOMBAT cr
.
```

The above run of CLEAN will result in the creation of a "clean" file WOMBAT.N from a "dirty" file WOMBAT.S.

15. CONCAT - Concatenate binary files

File name:	CONCAT.O
Description:	Concatenates two or more binary files.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	159 contiguous bytes.
Commands:	<ESC> exit to system monitor (during filename entry).

15.1 Description

Normal ASCII text files can be concatenated by using the COPY command in PIP (see section 5.5.1 of Part 2 of this Handbook - 2.465(82)) but binary files cannot be concatenated by this method because the ETX character (ASCII 03) is taken to be a terminator for all but the last file in the list (even if the extension is .C). CONCAT allows the concatenation of binary files; the number of sectors in the output file will always be the sum of the numbers of sectors in the input files. If the "length" of an input file is not considered to be an exact multiple of 126 bytes (i.e. an exact number of sectors) then up to 125 NUL characters will be inserted between the "end" of that file and the beginning of the next.

15.2 Preliminaries

Load CONCAT and execute it at its load address. Insert the disk containing the source files to be concatenated into one of the disk drives (normally DK0:). If the output file is to be written to another disk, insert the destination disk into the other drive.

15.3 User Input

Reply to the prompts as follows:

List files: Enter the list of files to be concatenated, separated by commas or spaces. The default extension is .C. Only disk files may be specified here.
<ESC> will exit to the system monitor.

Destination: Enter a name for the output file. The default extension is .C. The output may be sent to a device other than disk.
<ESC> will exit to the system monitor.

15.4 User Output

None.

15.5 Example

```
Binary File Concatenate utility  V1.0
List files: FILE1,FILE2,FILE3 cr
Destination: FIL123 cr
.
```

The above run of CONCAT will concatenate files FILE1.C, FILE2.C and FILE3.C and write the output to file FIL123.C.

16. VIDIT - Screen Editor

Note that the following description applies to Version 4.0 of VIDIT. Not all the facilities described are available in earlier versions.

File name:	VIDIT.O
Description:	Allows on-screen text editing.
File Format:	"Intel" object code as per 2.465(82) Appendix 3.
File Type:	Executable machine code utility.
Load Method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	All available memory is used, dependent on length of file being edited.

Commands:

<COPY>	Copy characters on the screen.
<F1>	Read to screen from a disk file.
<F2>	Write from screen to a disk file.
<F3>	Open or delete a disk file.
<F4>	Close a disk file.
<F5>	Scroll through to end-of-file.
<F6>	Scroll back to top of buffer.
<F7>	Clear a tab stop.
<F8>	Delete character with "wraparound".
<F9>	Insert a space with "wraparound".
<DEL LINE>	Delete the line containing the cursor.
<INSRT LINE>	Insert line below the current line.
<DEL CHAR>	Delete character at cursor position.
<INSRT CHAR>	Insert a space at cursor position.
<ROLL UP>	Scroll the screen up one line.
<ROLL DOWN>	Scroll the screen down one line.
<NEXT PAGE>	Scroll the screen up 23 lines.
<PREV PAGE>	Scroll the screen down 23 lines.
<PRINT>	Swap case of character at cursor.
<CLEAR SCRN>	Clear from cursor to end of screen.
<CLEAR LINE>	Clear from cursor to end of line.
<SO>*	Set bit 7 of subsequent characters.
<SI>	Reset bit 7 of subsequent characters.
<SET TAB>	Set a tab stop at cursor position.
<TAB>	Move to next tab stop position.
<ESC>	Display next letter as control symbol.
<ESC> <ESC>	Exit editor.

*Note <SO> is obtained by holding down SHIFT and pressing SO/SI.

16.1 Description

VIDIT allows the generation and subsequent editing of a file, of length limited only by the available space on a floppy disk. The various commands allow the user to alter the text that is currently being displayed, such as inserting new lines or deleting words that are not required. Text that has been rolled off the top of the screen can be recalled and altered, provided that it has not already been stored on disk, such as might be the case if the file is very long.

VIDIT may also be used to produce text files, with suitably

embedded commands, for later processing by a text formatting program such as TXTPRO, described in section 50 of this document.

16.2 Preliminaries

Load VIDIT and execute it at the load address. If a new file is to be created, insert in the drive the disk which is to receive the file. If an existing file is to be edited, insert in the drive the disk containing the file. The output of VIDIT may be stored on another disk, in which case insert the destination disk into the other drive.

16.3 User Input

If a new file is to be created, reply to the prompts as follows:

EDIT FROM?	Reply <RETURN> - The file is assumed to originate from the input device, i.e. keyboard.
BINARY MODE?	Normally reply <N>, <RETURN> is not required. Reply <Y> only if the file being created must contain the NUL (00) or ETX (03) characters, or if lines must not be terminated by CR, LF. It is possible using VIDIT to create a file containing any or all of the 256 possible 8-bit characters (see 16.5.27).
EXPAND TABS?	This prompt is produced only if you reply <N> to "BINARY MODE?". Reply <N> or <Y>, <RETURN> is not required. Reply <Y> only if you wish the output file to contain TAB characters (see below).
SHRINK OUTPUT?	This prompt is produced only if you reply <Y> to "EXPAND TABS?". Reply <N> or <Y>, <RETURN> is not required. If you reply <Y> the output file will contain TAB characters rather than spaces, corresponding to the tab stop positions.
TO?	Enter the name of the new file to be created, followed by <RETURN>. The default drive is DK0: and the default extension .S.

The screen will go blank so that creation of the file may now proceed. When entering text from the keyboard, a "wraparound" feature is incorporated whereby it is not necessary for the user to type <RETURN> at the end of each line. If text is entered as if onto one very long line, VIDIT will automatically split the text into separate lines without breaking up words (i.e. the splits will occur at the spaces between words). This facility is disabled when in binary mode.

If editing of an existing file is to be undertaken, reply to the prompts as follows:

EDIT FROM? Enter the name of the source file, followed by <RETURN>. The default drive is DK0: and the default extension .S. A list of filenames, separated by commas, may be entered in which case the files will be concatenated when read into VIDIT. Concatenation cannot be used if "binary mode" is specified.

BINARY MODE? Reply <N> or <Y>, <RETURN> is not required. If you reply <Y>, all characters in the file, including all control characters, will be displayed and can be edited (for the representation of control characters on the screen, see Appendix 1).

EXPAND TABS? This prompt is produced only if you reply <N> to "BINARY MODE?". Reply <N> or <Y>, <RETURN> is not required. Reply <Y> if you wish TAB characters (09) in the input file to be expanded to spaces, corresponding to the tab stop positions.

SHRINK OUTPUT? This prompt is produced only if you reply <Y> to "EXPAND TABS?". Reply <N> or <Y>, <RETURN> is not required. If you reply <Y>, tab characters will be substituted for spaces in the output file as appropriate. Note that because VIDIT first expands tab characters and then re-inserts them, it may not always put back tabs in exactly the same places as in the input file. It will, however, preserve the layout of the file as it appears on the screen.

TO? Enter the name of the edited output file, followed by <RETURN>. The default drive is DK0: and the default extension .S. If <RETURN> alone is typed, the filename will default to the same filename as that entered after "EDIT FROM?", but with an extension of .N, and the drive will default to DK0:

The screen will go blank. This should normally be followed by pressing <NEXT PAGE> or <ROLL UP> to transfer the first page or line of text from the source file onto the screen. Editing may then proceed.

16.4 User Output

When creation of a new file or editing of an existing file is complete, the user may store the result on disk using the following procedure:

Press <F5> to ensure that all of the file is rolled off the top of the screen. If this is not done, any text remaining on the screen, or text which has not yet been scrolled onto the screen, will not be written to the output file and will therefore be lost.

Press <ESC> TWICE to write all the information rolled off the

top of the screen to the new file; reply <Y> to the prompt "Exit? (Y/N)". When the file has been written to disk, VIDIT will exit to the monitor and return control to the user. If you reply <N> to "Exit? (Y/N)" you may continue editing by recalling the file onto the screen (<PREV PAGE>), although the beginning of a long file may already have been written to disk and will therefore be inaccessible.

16.5 Commands

This section deals with each command in turn, as shown in the command summary, in greater detail together with an example of its use. If the command key (or any key) is held down for longer than about 0.5 seconds, then the character or command will be repeated until the key is released, for example:

If <INSRT CHAR> is held down, then multiple spaces will appear at, and to the right of, the cursor position to allow whole words to be inserted.

The example text given is shown in upper case for emphasis only; a mixture of both upper and lower case characters is allowed with any of the commands.

16.5.1 <COPY> Copy characters on the screen.

This command, used in conjunction with the cursor control keys, allows any combination of characters on the screen to be copied so as to form a new line of text elsewhere on the screen. When <COPY> is pressed, an inverse video "block" will appear at the cursor position; this is the "write cursor". For the duration of the COPY command the normal flashing cursor is the "read cursor". The read cursor can be positioned using the normal cursor-control keys; the write cursor is advanced one position as each character is written (either as a result of normal key presses or by using the COPY key). If the <COPY> key is now pressed, the character at the read cursor position will be copied to the write cursor position, just as if it had been typed at the keyboard. Characters will continue to be copied each time the <COPY> key is pressed until the command is terminated by pressing the <RETURN> key. Whilst in the copy mode, the flashing cursor may be moved around to copy other sections of text elsewhere on the screen to the new line of text.

Example:

```
THIS IS LINE ONE OF TEXT
THIS IS LINE TWO OF TEXT
THIS IS LINE THREE OF TEXT
THIS WILL BE THE COPIED LINE
THIS IS LINE FIVE OF TEXT
THIS IS LINE SIX OF TEXT. #
```

The # symbol represents the block "write" cursor; this

is where the copied text will appear. The flashing "read" cursor has been moved to the position shown using the cursor control keys. If <COPY> is now pressed a number of times (or held down for greater than 0.5 sec), then the following will be the result:

```
THIS IS LINE ONE OF TEXT
THIS IS LINE TWO OF TEXT
THIS IS LINE THREE OF TEXT
THIS WILL BE THE COPIED LINE
THIS IS LINE FIVE OF TEXT
THIS IS LINE SIX OF TEXT. THIS WILL BE T#
```

This represents the condition in the middle of a copy; when the required copy has been completed the <RETURN> key is used to cancel the command. If the text to be copied exceeds 80 characters then the <COPY> key can be used for as long as necessary to complete the copy operation, again cancelling using the <RETURN> key:

```
THIS IS LINE ONE OF TEXT
THIS IS LINE TWO OF TEXT
THIS IS LINE THREE OF TEXT
THIS WILL BE THE COPIED LINE
THIS IS LINE FIVE OF TEXT
THIS IS LINE SIX OF TEXT. THIS WILL BE THE COPIED LINE
_
```

Characters copied using the COPY command take no account of whether the original text was in inverse video. Therefore the <SO> or <SI> commands should be used to determine the sense of bit 7, as with keyboard input. In particular, the COPY facility can be used to change a block of text from normal video to inverse video, or vice versa, without actually moving the text. This is achieved by positioning the cursor at the first character of the text and holding down the <COPY> key until it repeats, having previously used <SI> or <SO> to determine whether the text will be "copied" in normal or inverse video.

16.5.2 <F1> Read to the screen from a disk file.

This facility allows the contents of either a named disk file, or VIDIT's default scratch file, to be read to the screen. If <F1> is pressed when the cursor is on a BLANK LINE, then disk 0 is searched for a file with the name $$$$$$.$; if found, this file is read to the screen at the cursor position (space is automatically made for the file, existing lines of text are not overwritten). Reading of a long file may be aborted by pressing ESC. If the cursor is not on a blank line, the contents of the line are taken to be a device/file description in the normal form; if a file of the given name is found, it is read to the screen; otherwise the command has no effect. In other words, to read a named file to the screen, position the cursor

at the beginning of a blank line, type in the name of the file and press <F1>:

Example:

DK1:JAB_

If <F1> is pressed, the file DK1:JAB.S will be read to the screen.

Caution: If a DOS error is caused because the contents of the line do not correspond to a valid file description (e.g. a non-existent device mnemonic) the edit session may be lost (see section 16.6).

16.5.3 <F2> Write from screen to a disk file.

<F2> outputs the screen line containing the cursor to a disk file, and then moves the cursor down one line (scrolling the screen if necessary). If no disk file is open, a file with the name $$$$$$.$ will first be opened on DK0: (if file $$$$$$.$ already exists, it is first deleted). A file may have been opened by a previous use of <F2> or by use of <F3>. When as many lines as required have been written to the file (with repeated use of <F2>) the file may be closed using <F4>, or may simply be recalled with <F1>, which automatically closes the file before reading it. This facility, in conjunction with <F1>, allows very easy manipulation of blocks of text (e.g. the exchanging of paragraphs).

Example:

THIS IS LINE ONE
THIS IS LINE TWO
THIS IS LINE THREE

If <F2> is pressed THREE TIMES, and <F4> is then pressed, these three lines will be written to a disk file whose name was determined by a previous <F3> command, or to file DK0:$$$$$$.$ if no <F3> command had been issued.

16.5.4 <F3> Delete and/or open a disk file for writing.

<F3> takes the contents of the line containing the cursor as a filename, in the usual format, and attempts to open an output file of that name. If a file of the given name already exists, it is first deleted. <F3> is usually used to open a file for subsequent writing with <F2>, but if <F3> is followed immediately by <F4> the effect is simply to delete the file.

Example:

WOMBAT.N _

If <F3> is pressed the file DK0:WOMBAT.N will first be deleted, if one existed, and a new file of that name will be opened for output.

Caution: As with <F1>, use of <F3> may give rise to a DOS error if the screen line cannot be interpreted as a valid filename (or, for example, the disk drive gate is not shut). In such circumstances the edit session may be lost. See section 16.6 for more details of recovering from error conditions.

16.5.5 <F4> Close a disk file.

If a disk file has been opened for output, through the action of <F3> and/or <F2>, it is closed. If no file has been opened, <F4> has no effect. <F4> is normally used only if two or more files are to be written to disk without being read back with <F1>, as <F1> closes the file itself.

16.5.6 <F5> Scroll until end-of-file

This command is used to scroll ALL the text of the file off the top of the screen, so that the <ESC> key, pressed twice, may be used to store the edited text on the disk before leaving VIDIT. The <F5> command may be used at any time with the cursor in any position. As with the <ROLL UP> command, any reading from the source file or writing to the destination file will slow the scroll function. This command may be used to access the end of a file quickly, by following it with <ROLL DOWN> or <PREV PAGE> to roll the end of the file from "above" the screen onto the display. Any text that had been written to the destination file when this was done will not be accessible.
ESC may be pressed to abort the <F5> command before the end-of-file is reached.

16.5.7 <F6> Scroll until top of buffer

This command is equivalent to repeated <PREV PAGE> commands, until the top of the RAM buffer is reached. If none of the file being edited has yet been written to the output file, <F6> will scroll right down to the beginning of the file. If some of the file has been written, <F6> will scroll until the first text still in the buffer is displayed.
ESC may be pressed to abort the <F6> command before the top of buffer is reached.

16.5.8 <F7> Clear tab stop

If the current cursor column corresponds to a tab stop position, then the tab stop is cleared. If no tab stop has been set at the cursor position, <F7> has no effect. The clear tab stop function can also be obtained by pressing CTRL/<SET TAB> (hold down CTRL and press <SET TAB>). See <SET TAB> for more information.

16.5.9 <F8> Delete character at cursor with wraparound

If <F8> is pressed the character at the cursor position will be erased and replaced by the character that was immediately to the right of the cursor before the command key was pressed. All characters to the right of the cursor to the end of the NEXT line will be moved left one place, the first character on the next line being moved to the last position on the line containing the cursor. The last character on the next line is replaced by a space.

Example:

THIS IS A LINE OF TEXT; HERE IS WHERE WE WILL DELETE
AND HERE IS THE NEXT LINE OF TEXT.

If the <F8> key is now pressed, with the cursor at the position shown, the following will be the result:

THIS IS A LINE OF TEXT; ERE IS WHERE WE WILL DELETE
ND HERE IS THE NEXT LINE OF TEXT.

Note that the "A" of "AND" has been moved to the end of the first line, but is not shown here.

16.5.10 <F9> Insert space at cursor with wraparound

If <F9> is pressed those characters at, and to the right of, the cursor position up to the end of the NEXT line, will be moved one space to the right. The last character on the line will be moved to the beginning of the next line, and the last character on the NEXT line will be lost. A gap equivalent to one character's width will appear at the cursor position in which a new character may be typed.

Example:

THIS IS A LNE OF TEXT
AND THIS IS THE NEXT LINE.

If the cursor is placed here and <F9> is pressed, the following will be the result, allowing the correct letter to be inserted at the cursor position:

THIS IS A L NE OF TEXT
AND THIS IS THE NEXT LINE.

Note the space before the "A" of "AND", which is assumed to have wrapped round from the first line.

16.5.11 <DEL LINE> Delete the line containing the cursor

If <DEL LINE> is pressed the line at the cursor position will be deleted and all lines of text below the cursor will be moved up by one line and the cursor will be placed at the beginning of the line. The next line of text "below" the screen will now become visible on the bottom line. All lines above the cursor position will be unaffected.

Example:

THIS IS LINE ONE OF TEXT
THIS IS LINE TWO OF TEXT
THIS IS LINE THREE OF TEXT
THIS IS LINE FOUR OF TEXT

If the <DEL LINE> key is now pressed, with the cursor at the position shown, the following will be the result:

THIS IS LINE ONE OF TEXT
THIS IS LINE THREE OF TEXT
THIS IS LINE FOUR OF TEXT

16.5.12 <INSRT LINE> Insert a line at the cursor position

If <INSRT LINE> is pressed the line containing the cursor, and all those on the screen above the cursor, will be moved up by one line and the cursor will be positioned at the start of the blank line. Lines below the cursor will remain unchanged. No text will be lost when this command is used.

Example:

THIS IS LINE ONE OF TEXT
THIS IS LINE TWO OF TEXT

If the cursor is placed anywhere on the first line of text, and the <INSRT LINE> key is pressed, the following will be the result, allowing a new line of text to be inserted:

THIS IS LINE ONE OF TEXT

THIS IS LINE TWO OF TEXT

16.5.13 <DEL CHAR> Delete character at cursor position

If the cursor is on a line of text and <DEL CHAR> is pressed, the character at the cursor position will be erased and replaced by the character that was immediately to the right of the cursor before the command key was pressed. All characters to the right of the cursor to the end of the line will be moved left one place, the last character on the line being replaced by a space.

Example:

THIS IS A LINE OF TEXT; HERE IS WHERE WE WILL DELETE

If the <DEL CHAR> key is now pressed, with the cursor at the position shown, the following will be the result:

THIS IS A LINE OF TEXT; ERE IS WHERE WE WILL DELETE

16.5.14 <INSRT CHAR> Insert a space at cursor position

If the cursor is on a line of text and <INSRT CHAR> is pressed, those characters at, and to the right of, the cursor position up to the end of the line, will be moved one space to the right. The last character on the line will be lost. A gap equivalent to one character's width will appear at the cursor position in which a new character may be typed.

Example:

THIS IS A LNE OF TEXT

If the cursor is placed here and <INSRT CHAR> is pressed, the following will be the result, allowing the correct letter to be inserted at the cursor position:

THIS IS A L NE OF TEXT

16.5.15 <ROLL UP> Scroll screen up one line

This command will roll all lines of text on the screen up by one line. The top line will be scrolled "off the screen" but will not be lost (unless the RAM buffer is full in which case the line may be sent directly to the output file) and the bottom line will be read from the RAM buffer or, if empty, the input file. If the input file is empty a blank line will be scrolled onto the screen.
<ROLL UP> normally takes a very short time, but if text must be read from the input file or written to the output file, a small delay will result.

16.5.16 <ROLL DOWN> Scroll screen down one line

This command will roll all lines of text on the screen down by one line. The bottom line on the screen will disappear "below" the screen, but will not be destroyed or lost. The line of text that was not visible, but "above" the screen before the <ROLL DOWN> key was pressed, will now appear at the top of the screen. If there is no data in the RAM buffer "above" the top of the screen, <ROLL DOWN> has no effect.
If the <ROLL DOWN> key is used to try and look at the start of the file, and some of the text has already been written to the output file (because the RAM buffer has filled), then it will not be found possible to roll back all the way to the start.
<ROLL DOWN> normally takes a very short time but occasionally a disk access may occur, giving rise to a small delay. This is a consequence of VIDIT's buffer management and is quite normal.

16.5.17 <NEXT PAGE> Scroll screen up 23 lines

This command has an identical effect to pressing <ROLL UP> 23 times. It may be used to skip a large number of lines rapidly or to display the first "page" of text from the disk when commencing an edit of an existing file (see section 16.3). <NEXT PAGE> deliberately scrolls one less line than the number of lines on the screen (24) so that, if used to read the first "page" of a file, a blank line is left above the first line of the file. This aids the insertion of new lines before the start of the file, should this be required.

16.5.18 <PREV PAGE> Scroll screen down 23 lines

This command has identical effect to pressing <ROLL DOWN> 23 times. If there are not 23 lines available "above" the screen then only the number of lines available will be rolled down. If <NEXT PAGE> is followed immediately by <PREV PAGE> the original screen contents will be restored, unless the RAM buffer was so full that some of the lines scrolled off the screen by <NEXT PAGE> were written to the output file.

16.5.19 <PRINT> Swap case of character at cursor position.

If the character at the cursor position is an alphabetic character (strictly: if its ASCII code is between 40H and 7FH) then it will be changed from lower-case to upper-case or vice versa; the cursor is then moved one place to the right. A block of text may be changed from lower-case to upper-case or from upper-case to lower-case by positioning the cursor under the first character and holding down <PRINT> until it repeats.

Example:

Here is some lower-case text

If the <PRINT> key is now pressed TEN times the following will be the result:

hERE IS SOme lower-case text

16.5.20 <CLEAR SCRN> Clear from cursor to end of screen

This command will destroy all text on the screen from the cursor position to the bottom right hand corner. Once this command has been issued then all the cleared text is lost and cannot be recovered. The cursor will remain in the same position as prior to the <CLEAR SCRN> key being pressed. Any text to the left and above the cursor, or "below" the screen, will be unchanged. If the cursor is in the top left hand position of the screen, then pressing <CLEAR SCRN> will destroy a complete "page" (screen full) of text.

16.5.21 <CLEAR LINE> Clear from cursor to end of line

When the cursor is at any position on a line of text, and this key is pressed, all the characters from the cursor position to the right hand side of the screen on the same line will be deleted. No other characters on any other line will be affected.

Example:

THIS IS A LINE OF TEXT; DELETION WILL BE FROM HERE

If <CLEAR LINE> is now pressed with the cursor at the position shown, the following will be the result:

THIS IS A LINE OF TEXT; D_

16.5.22 <SO> Set bit 7 (display in inverse video)

This command, implemented by holding the <SHIFT> key and pressing SO/SI, will set bit 7 of all subsequent characters typed in. This will be indicated by the characters appearing on the screen in inverse video (dark characters on a light background). This facility may be required when editing a "binary" file or can, for example, allow access to the graphics printing set of certain printers, such as the EPSON MX-80. Termination of this command is achieved by pressing the same key SO/SI, but without holding the <SHIFT> key down.

16.5.23 <SI> Reset bit 7 (normal display)

This command will cancel the effect of <SO>, and is implemented by pressing the SO/SI key, but without holding the <SHIFT> key down. All text will now appear in its normal mode as light characters on a dark background.

16.5.24 <SET TAB> Set a tab stop at the cursor position

When first loaded, VIDIT is set up to have nine tab stop positions, at columns 8, 16, 24, 32, 40, 48, 56, 64 and 72 (the left-most column is column 0). Additional tab stop positions, up to a total of 19, may be set up by positioning the cursor in the appropriate column and pressing <SET TAB>. If a tab stop has already been set at this column, or if 19 tab stops are already active, <SET TAB> has no effect.

It is possible to save a version of VIDIT which is customised with different default tab stop positions. Simply load and run VIDIT from your system disk, set up the new tab stop positions (using <CLEAR TAB> and <SET TAB>), exit VIDIT (press ESC twice) and save the new version to disk (address range 0,7FF).

16.5.25 <TAB> Move cursor to next tab stop position

This command will move the cursor to the next tab stop position, or if there are no more tab stops on the current line the cursor will be moved to the beginning of the next line. <TAB> is equivalent to pressing the cursor right key the appropriate number of times, it does NOT insert a tab character (09) into the output file (although one may be inserted by VIDIT if the SHRINK OUTPUT option has been selected - see 16.3).

16.5.26 Cursor Control Keys:

Cursor Left (same as <BACKSPACE>)
Cursor Right
Cursor Down (same as <LINE FEED>)
Cursor Up
Cursor Home
Cursor to beginning of line / cancel COPY (<RETURN>)

The cursor control keys and the <RETURN> key allow the user to position the flashing underline cursor at any position on the screen without destroying any existing text. The cursor indicates where the next printing character will be written, where the next command will take effect, or where characters will be copied from when in COPY mode.

The cursor left key or <BACKSPACE> will move the cursor one character to the left of its current position. If the cursor is already on the extreme left, then pressing this key again will move it to the last position on the previous line. If the cursor is in the top left hand corner of the screen, then pressing this key will cause the screen to scroll down (unless at the top-of-buffer).

The cursor right key will move the cursor one character to the right of its current position. If the cursor is already on the extreme right, then pressing this key will move it to the first position on the next line. If the cursor is in the bottom right hand corner of the screen, then pressing this key will cause the screen to scroll up one line. The cursor right key must not be confused with the space bar in its operation. If the space bar is pressed when there are no characters present at the cursor position, the result will be as though the cursor right key had been pressed. However, if there are characters present, and the space bar is pressed, then the original characters will be deleted and replaced by a space.

The cursor down key or <LINE FEED> will move the cursor one line down the screen from its current position. If the cursor is already at the bottom of the screen, then pressing this key will cause the screen to scroll up one line.

The cursor up key will move the cursor one line up the screen from its current position. If the cursor is already at the top of the screen, then pressing this key will cause the screen to scroll down one line (unless already at the top-of-buffer).

The cursor home key will move the cursor from its current position to the top left hand corner of the screen. If the cursor is already at the top left position, then pressing this key will have no effect.

The <RETURN> key will move the cursor from its current position to the first position on the same line. If the cursor is already on the extreme left, then pressing this key will cause no cursor movement. If the COPY command is in operation when the <RETURN> is pressed, then the command will be terminated. (See section 16.5.1 for a description of the <COPY> key command).

16.5.27 <ESC> (PRESSED ONCE) Display next as control symbol

When <ESC> is pressed once, the next printing character typed will be displayed as a control symbol or graphics character. The control symbols and graphics characters are the same as those produced by CMPARE and DISFUN and are shown in Appendix 1 of this handbook. They correspond to ASCII values in the range 0 to 1F (hex) or, if in inverse video, 80 to 9F (hex).

Note that, unless VIDIT is operating in "binary mode" some control characters may cause strange effects if they are scrolled off the top or bottom of the screen and back again. In particular, the carriage return (ESC M) and ETX (ESC C) symbols may have unwanted effects. NUL (ESC @) and line feed (ESC J) characters are discarded if scrolled off the screen and back again.

If "binary mode" has been selected, VIDIT allows the entry, display and editing of files containing any or all of the 256 possible 8-bit character codes. The only control code to cause special action in VIDIT is CR (0D hex, ESC M) which, as well as being displayed as the appropriate symbol (see Appendix 1) also causes an "end of line" action, i.e. the rest of the screen line is left blank. This convention is the same as that adopted by CMPARE and DISFUN, and is convenient in allowing text files to be examined in binary mode whilst preserving the general layout of the file on the screen. It should be noted that when the last NON-SPACE character on a line is CR, then the rest of the line is NOT treated as part of the (binary) file. If, however, the last non-space character is not CR, the line is taken as exactly 80 characters of binary data. If you delete the terminating CR on a line, this will also have the effect of adding the trailing spaces on

that line to the file.

16.5.28 <ESC> (PRESSED TWICE) Exit editor; save file

This command is used in conjunction with the <F5> command in section 16.5.6 above. Everything that has been rolled off the top of the screen will be transferred to the output file. Any text either still visible or "below" the the screen will be lost when this command is used (see 16.4). When storage is completed the screen is cleared and control is transferred back to the monitor (full stop prompt).
This command requests confirmation by displaying "Exit? (Y/N)" on the bottom line of the screen, to which the user must respond "Y" (in upper or lower case) to cause the command to complete. Any other response causes the command to be aborted (the bottom line is restored).

16.6 Recovering from DOS errors

If a DOS error occurs (i.e. one of those errors listed in Appendix 1 of Part 2 of this handbook - 2.465(82)) the normal error message (in inverse video) will be displayed on the screen. However, unless the error occurred after issuing the <ESC> <ESC> command, the usual action of aborting to PIP is suppressed and instead control is returned to VIDIT. This allows editing to continue but, depending on the error which occurred, it may not be possible to save the edited text to disk!

If the error condition is "recoverable" in being caused by something incidental to the main editing operation (e.g. an invalid filename was used for the <F3> command - see 16.5.4) then whether or not editing can be completed depends on whether anything has yet been written to the output file. If it has, then it will not be possible to save the edited text because the output "channel" to the disk will have been aborted (a "NO OPEN FILE" error will result). In such circumstances you should follow the "panic" procedure outlined below. If nothing has yet been written, it should be possible to finish the edit session successfully, although it will be necessary to re-type the text which has been overwritten by the error message.

If the error condition is "irrecoverable" (e.g. a DISK FULL error on writing the output file or a CRC ERROR in the input file) then the following "panic" procedure should be followed (if the effort is justified):

1. If anything has been written to the output file, remove the destination disk and put it to one side.

2. Scroll any text which remains on the screen off the top of the screen, if possible, or failing that off the bottom of the screen. Exit VIDIT with <ESC> <ESC>. In some circumstances you may find it impossible to exit VIDIT

other than by a system reset (CTRL/BREAK)!

3. Dump the contents of VIDIT's text buffer RAM to a disk file (NOT on your destination disk) by using the monitor's W command:

```
.W 800,6C52 cr
WRITE BINARY FILE? PANIC.1 cr
NEXT BLOCK cr
.
```

4. If something had been written to the original destination file, and you consider it worth recovering, try the following: Place the disk in drive 0, place a "scratch" disk in drive 1, enter the "Mostek" editor (E :ED) and issue the following commands:

```
.E :ED cr
FROM? @@@@@@.@ cr
TO? DK1:PANIC.2 cr
>R cr
>B Q cr
>esc
.
```

The symbols shown here as @ should in fact be entered as CTRL/@, i.e. hold down <CTRL> and type @, and will not echo to the screen. With luck, this command will make a copy of the partially written file which, never having been closed, will not appear in the disk directory. If the file is longer than the editor buffer (BUFFER FULL message) then use subsequent Y, R and Q commands to transfer the whole file. If a DOS error occurs on reading the file (probably ILLEGAL TRACK NUMBER) then re-enter the editor (E :ER) and write the buffer contents to disk.

5. You should now have either two or three disk files: The original (unedited) file, a "RAM dump" file and possibly a recovered "destination" file. By combining sections of these files you should be able to create a new file with most of the edited text intact. The easiest way to combine the files is to use VIDIT and concatenate the files on entry:

```
EDIT FROM? OLDFILE,PANIC.1,PANIC.2 cr
```

17. LOOK - Memory Search

File name:	LOOK.O
Description:	Searches the memory for a specified byte pattern.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	512 contiguous bytes.
Commands:	<ESC> exit to system monitor (during search string entry)

17.1 Description

LOOK will search a specified area of memory, listing all the addresses at which a certain sequence of bytes occurs. The bytes can be specified in either hex or character form. Applications include locating jumps, calls etc to specific addresses or locating text strings.

17.2 Preliminaries

Load LOOK outside of the area of memory to be searched (the program occupies about 200H bytes) and execute at the load address. Note that the linking loader uses an area of memory as scratchpad (see section 3.1) and care should be taken to ensure that this does not corrupt the data to be searched. If the data to be searched is loaded from a disk file, it is safer to load LOOK before loading the data.

17.3 User Input

Reply to the prompts as follows:

LIST BYTES AS HEX PAIRS OR 'ASCII':	Enter the wanted bytes in sequence, separated by spaces or commas, and terminated by <RETURN>. Each byte can be specified either as a hex value (one or two hex digits) or as ASCII code(s) by means of a character string between quotes ('). A maximum of 16 bytes is allowed - a longer sequence will be truncated to this number. <ESC> returns control to the monitor.
ADDRESS RANGE (START,END)?	Enter the addresses between which the search is required, in hex, separated by a comma or space and terminated by <RETURN>. The entry format is the same as for monitor commands, e.g. only the last four digits of the address are significant.

17.4 User Output

Each sequence found is listed on a new line, in the form:
<Address> <Hex byte sequence> <Character sequence>

If the sequence cannot be found, the message NO MATCH will be displayed.

17.5 Example

```
.L 0 cr
LOAD FROM? LOOK cr
BEG ADDR 0000
EXECUTE  0000
END ADDR 0164
UNDEF SYM 00
#E cr

LIST BYTES AS HEX PAIRS OR 'ASCII': 'FILE' cr
ADDRESS RANGE (START,END)? C000,FBFF cr

EB69  46 49 4C 45                        FILE
ECD8  46 49 4C 45                        FILE
F010  46 49 4C 45                        FILE
F9F7  46 49 4C 45                        FILE

LIST BYTES AS HEX PAIRS OR 'ASCII': 0D,0A,03 cr
ADDRESS RANGE (START,END)? C000,FBFF cr

DA47  0D 0A 03                           ...
DA55  0D 0A 03                           ...
E411  0D 0A 03                           ...

LIST BYTES AS HEX PAIRS OR 'ASCII': esc
.
```

18. INIT - Initialise disk directory

File name: INIT.O
Description: Initialises the directory on a disk.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: 266 contiguous bytes.

18.1 Description

A new, formatted, disk must be initialised before it can be used on Zelda. Initialisation consists of writing an empty "directory" on the disk and also storing on the first sector of track zero a text string identifying the owner of the disk. This process need only be carried out once for each disk. INIT may also be used as a "delete all" facility if carried out on a disk on which files have previously been written.
Because it is intended that INIT be customised to include the details of the particular Zelda owner, the source code for the Designs Department version (INITDD.S) is included on the utilities disk. This may be edited so as to incorporate the name and address of the appropriate department. Note that only one sector (126 bytes) is available for this information so a text string longer than this will be truncated. DSKED (see section 20) will allow you to examine what has been stored on the disk.

18.2 Preliminaries

Load INIT (normally at 0) and execute it at the load address. Insert the disk to be initialised into drive 0.

18.3 User Input

Reply to the prompt as follows:

CONFIRM INITIALISE DISK? Type "Y" to initialise the disk; any other character will cause the operation to be aborted. <RETURN> is not required.

18.4 User Output

None.

18.5 Example

```
CAUTION: THIS ROUTINE DELETES ALL FILES
CONFIRM INITIALISE DISK? Y_
.
```

19. ANALYZ - Analyse disk contents

File name:	ANALYZ.O
Description:	Checks that all sectors on the disk are readable and constructs a "pseudo directory" of the files on the disk.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	5098 contiguous bytes.

19.1 Description

ANALYZ measures the time taken for a disk drive to reach its proper rotational speed, reads all sectors on a disk and reports any errors found, and optionally creates a "pseudo directory" for the disk by tracing the linked-lists of sectors of which the files are comprised. ANALYZ is principally of use as a "verify" routine, in that it will report the track and sector number of any sectors which cannot be read, but can also be used as an "undelete" or "recover" program in certain circumstances, such as a major corruption of the disk's directory.

When generating the "pseudo directory" ANALYZ cannot, of course, know what the original file names were but it can attempt to find where files were situated on the disk (even if deleted) and create a new directory corresponding to these "files". This procedure will fail if the disk appears to contain more than 99 files, or if the files are so fragmented that the pseudo directory becomes too big. If tracing of the linked-lists indicates that an area of the disk belongs to two or more "files" (i.e. two links are found to point to the same sector) then ANALYZ will arbitrarily assign this area to one of the files. However, when the files are read, it will not be apparent to the user to which file ANALYZ has assigned the area as files are read only by following links - not by reference to the directory. Subsequently deleting one of the files, and then writing new data to the disk, may result in the other "file" being corrupted. It is suggested, therefore, that if ANALYZ has been run to recover a deleted file or to regenerate a directory which has become damaged, then the wanted file(s) are copied onto another disk and the ANALYZed disk is re-initialised.

19.2 Preliminaries

Load ANALYZ and execute it at the load address. Insert the disk to be ANALYZed into drive 0 and press <RETURN>.

19.3 User Input

Reply to the prompt CONSTRUCT PSEUDO-DIRECTORY? with "Y" if you want ANALYZ to attempt this function; <RETURN> is not required. Any other character will exit to the monitor. When ANALYZ has constructed the pseudo-directory, it will automatically execute PIP's LIST command to display the directory to the user (see section 5.5.4 of Part 2 of this Handbook - 2.265(82)). If at this point an error message is produced, or if the directory displayed is in fact the directory on the disk rather than the pseudo-directory, then ANALYZ has failed for one of the reasons mentioned above.

If a pseudo-directory has successfully been created and displayed, it is resident in the system RAM but NOT on the disk itself. Any error condition (e.g. an illegal PIP command or a COPY command being aborted by CTRL/C) will result in the directory being marked invalid; ANALYZ will have to be run again to re-create it. If several files need to be recovered, it may be convenient to store the pseudo-directory on the disk. This may be achieved by issuing the following PIP commands immediately after the display of the pseudo-directory (press <ENTER> twice to enter PIP):

```
*RENAME .Z=.S cr
*RENAME .S=.Z cr
```

Do NOT do this unless you are certain that there is no information in the original disk's directory which you need to preserve (e.g. file names)! Of course, if the disk directory is corrupted in such a way that it cannot be written to (e.g. a CANNOT FIND SECTOR error at sectors 1 to 13 on track 1) then it will not be possible to write the pseudo-directory to the disk in this way. A possibility in this case is to run MFT immediately following ANALYZ to copy several files onto another disk; drive 0 must not be opened as this will also mark the RAM directory as invalid, so load MFT from drive 1. Of course, if one of the files you are trying to copy contains a disk error the operation will be aborted and ANALYZ will have to be run again.

19.4 User Output

The following messages may be produced by ANALYZ; in all cases the sector numbers are given in decimal from 1 to 26 and the track numbers in decimal from 0 to 76:

DISK BECAME READY IN n.nn SECONDS
The time given is the approximate time between the user pressing <RETURN> and the disk drive's "ready" line becoming active. This should be about 2 seconds in the case of the Caldisk drives supplied with Zelda.

DISK FAILED TO BECOME READY WITHIN 2.56 SECONDS
If this message is produced, ANALYZ is aborted and control is returned to the monitor. The usual cause is the drive gate being open or no disk inserted.

CRC ERROR AT SECTOR nn OF TRACK mm
This indicates a CRC error at the given sector. No indication is given of whether the CRC error is in the sector I.D. or in the data field of the sector, although it is more likely to be in the data field.

LOST DATA AT SECTOR nn OF TRACK mm
This indicates a "lost data" condition signalled by the disk controller. Unless caused by a hardware fault on the equipment, this suggests a severely corrupted disk or an unsuitable disk format (e.g. 256 byte sectors).

CANNOT FIND SECTOR nn OF TRACK mm
This indicates that the sector I.D. of the given sector was not readable. In this case the disk is either severely corrupted or is not correctly formatted for use on Zelda (e.g. double density).

DISK NOT READY - TEST ABORTED AT SECTOR nn OF TRACK mm
This message is produced only infrequently and indicates that the disk became "non-ready" after the tests were commenced. This is probably because the drive gate was opened whilst the tests were in progress, or else indicates a drive fault.

19.5 Example

```
        --- FLOPPY DISK ANALYSIS ---

INSERT DISK IN DRIVE 0 AND PRESS <RETURN> cr

DISK BECAME READY IN 2.07 SECONDS

TEST IN PROGRESS:

CRC ERROR AT SECTOR 20 OF TRACK 70
CANNOT FIND SECTOR 20 OF TRACK 71
CONSTRUCT PSEUDO-DIRECTORY? Y
DK0:FILE01.S  693 SECTORS
DK0:FILE02.S   29 SECTORS
DK0:FILE03.S   16 SECTORS
DK0:FILE04.S   52 SECTORS
DK0:FILE05.S   79 SECTORS
DK0:FILE06.S   17 SECTORS
DK0:FILE07.S  536 SECTORS
DK0:FILE08.S    2 SECTORS
DK0:FILE09.S  409 SECTORS
DK0:FILE10.S   34 SECTORS
DK0:FILE11.S    1 SECTORS
    Unused     95 SECTORS

.
```

20. DSKED - Examine/edit disk sectors

File name:	DSKED.O
Description:	Displays the contents of disk sectors, and allows them to be edited.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	2300 contiguous bytes, including sector buffer and stack.
Commands:	<CR> Read next sector
	- Read previous sector
	D Change the currently selected disk drive.
	E Edit the current sector in RAM.
	F Fill the current sector with a data byte.
	Q Exit to the monitor.
	R Re-read the current sector.
	S Read a specified sector.
	W Write the current sector to disk.

20.1 Description

DSKED allows the contents of disk sectors to be displayed, and edited if required. Such low-level access to the disk is usually necessary only if the disk is not in Zelda format (e.g. a CP/M disk) or in the event of a corrupted disk directory. DSKED should be used with care because changes to any of the directory sectors on the disk, or to the link bytes at the end of each sector, may render one or more files (or the whole disk) unreadable by the normal Zelda utilities.

20.2 Preliminaries

Load DSKED (normally at 0). Insert the disk you wish to examine/edit (usually into drive 0) and execute DSKED at the load address. DSKED initially selects the drive which was last accessed, i.e. the drive from which DSKED itself was read.

20.3 User Input

There is no initial dialogue with the user. DSKED reads and displays the first sector on the disk (sector 1 on track 0) and then prompts the user for a command.

20.4 User Output

DSKED displays at the top of the screen a summary of disk parameters: drive number, number of tracks, minimum and maximum sector numbers (on each track), current track number, current sector number and logical sector number. The logical sector number is a decimal number which starts from 0 (the first sector on track 0) to 2001 (the last sector on track 76). The Zelda Disk Operating System works in logical sector numbers, and the link bytes at the end of each sector correspond to the logical number of the next sector in the file.

The contents of the current sector are displayed both in hexadecimal and ASCII form, in a similar way to the monitor's M command. A full point '.' is displayed rather than an ASCII character if the data value is less than 20 (hex) or greater than 7F (hex).

If an invalid command letter is typed (i.e. not one of <CR>, -, D, E, F, Q, R, S or W) DSKED displays a list of the valid commands at the bottom of the screen.

20.5 Commands

One of the following command letters may be typed in response to the prompt "Command?". Either upper-case or lower-case may be used:

<CR> Read the next sector.
The current sector number is incremented, the sector is read from the disk and the contents of the sector are displayed. If the logical sector number is 2001 when the command is issued, the message "Reached inside limit of disk" is produced and the sector number is left unchanged. If a disk error occurs (e.g. CRC ERROR) then control is retained by DSKED, but the displayed data are probably incorrect.

\- Read the previous sector.
The current sector number is decremented, the sector is read from the disk and the contents of the sector are displayed. If the logical sector number is 0 when the command is issued, the message "Reached outside limit of disk" is produced and the sector number is left unchanged. If a disk error occurs (e.g. CRC ERROR) then control is retained by DSKED, but the displayed data are probably incorrect.

D Change the selected disk drive.
The prompt "Disk (0/1)?" is displayed, and the appropriate key should be pressed. <CTRL>/C will cause the command to be aborted. If the drive number entered is neither 0 nor 1, the message "Bad disk" is produced. When the new drive has been selected, the current sector is read from that drive and displayed.

E Edit sector contents (in RAM).
The message "Edit in progress." is displayed and the cursor is moved to the first data item in the sector. The sector contents may be changed simply by overwriting either the hexadecimal or the ASCII parts of the display. When in edit mode, the following control keys are active:

<LEFT>	Move cursor left one byte.
<RIGHT>	Move cursor right one byte.
<UP>	Move cursor up one line (16 bytes).
<DOWN>	Move cursor down one line (16 bytes).
<ESC>	Swap cursor between hex and ASCII fields.
<RETURN>	Exit edit mode.

The Edit command affects only the data stored in RAM; to update the disk the Write command must be used.

F Fill buffer with specfied value.
The prompt "Fill buffer with?" is produced, in response to which a hexadecimal value should be entered, followed by <RETURN>. Alternatively, <ESC> may be pressed, resulting in the prompt "Character?"; any character typed will be used to fill the buffer. <CTRL>/C will abort the command. The Fill command affects only the data stored in RAM; to update the disk the Write command must be used.

Q Quit to the monitor.
Control is immediately returned to the monitor (full point prompt).

R Re-read the current sector.
This command can be used to re-read the current sector to RAM following any alterations to the RAM contents (using the Edit or Fill commands).

S Set sector number.
The prompt "Sector?" is displayed, in reponse to which a decimal sector number (in the range 0-2001) should be entered, followed by <RETURN>. Alternatively, a hexadecimal value may be entered if preceded by an ampersand ("&"). <CTRL>/C will abort the command. The current sector number is updated, the sector is read from the disk and its contents are displayed. If the sector number is out of range the message "Bad sector" is produced.

W Write sector to disk.
The prompt "Write current sector. Are you sure (Y/N)?" is displayed. If "Y" is entered (in upper or lower case) the current RAM contents are written to the disk; any other character aborts the command.

21. RAMTST - RAM Test

File name:	RAMTST.O
Description:	Performs system RAM test and reports faults found.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	0000; relocatable.
Execute address:	Same as load address.
Space occupied:	208 contiguous bytes (all 32,768 bytes of the main RAM are affected).

21.1 Description

RAMTST performs a thorough test of the 32 Kbytes of system RAM on the UN27/13, and displays the address of any faulty locations. The test is performed repetitively until the system is reset: there is no other exit from this utility. The contents of the tested memory are destroyed by the test.

The program fills the memory with a known data pattern, then reads the memory and reports any errors found. The value written into each location is:

(H + L + B + (H + L) DIV 256) MOD 256

where H = address DIV 256
L = address MOD 256
B = an 8-bit counter incremented each time the pattern is written into memory.

RAMTST is particularly useful for finding intermittent faults in memory units, or for checking a suspect unit which the system self-test reports as normal. The system RAM must be functioning sufficiently well for the program to be loaded from disk; also it should be realised that RAMTST itself resides in the memory unit being tested. The memory is tested from the first address above the program space up to 7FFFH, so selected parts of the memory may be tested by loading the utility at an address other than 0.

21.2 Preliminaries

Load RAMTST (normally at 0) and execute it at the load address.

21.3 User Input

None.

21.4 User Output

If no RAM faults are detected the current "cycle number" (B in the formula given in 21.1 above) is displayed (in hexadecimal) on the screen. This should continue to increment (in modulo 256 fashion) indefinitely.

Where the contents of a memory location as read do not match the value written into that location, the fault is reported as:

RAM failed at AAAA - data should be BB but was CC (Write error)

where AAAA = address at which the fault was found
BB = data written to that address
CC = data read from that address

All values are given in hexadecimal. RAMTST makes an attempt to determine whether the fault is a read error or a write error by re-reading from the faulty location and comparing the data read with both the original data written and the data read when the error was detected. If the data now reads as correct, the indication "Read error" is given. If the data reads as the same incorrect value as before, "Write error" is reported. If neither condition applies, "Unknown error" is indicated.

21.5 Commands

There are no keyboard commands. The system must be reset (CTRL/BREAK) to terminate the program. If many faults are found it may be useful to WAIT the system by using the switch on the UN26/31 to allow time to examine the display.

21.6 Example

```
.L 0 cr
LOAD FROM? RAMTST cr
BEG ADDR 0000
EXECUTE  0000
END ADDR 00CF
UNDEF SYM 00
#E cr
00: 01: 02: 03: 04: 05:
RAM failed at 1000 - data should be 10 but was 90 (Write error)
RAM failed at 1000 - data should be 11 but was 91 (Write error)
```

22. RDCAS - Read data from serial input

File name: RDCAS.O
Description: Reads data from a serial input, buffers it and writes it to an output device.
File format: "Intel" object code as per 2.465(82) Appendix 3.
File type: Executable machine code utility.
Load method: Monitor L command.
Load address: 0000; relocatable.
Execute address: Same as load address.
Space occupied: All available memory is used for buffering.

22.1 Description

RDCAS is specifically intended for reading from a device which supplies data at a fixed rate, with no provision for handshaking. Typically such a device will be a tape cassette (via suitable modem) or a serial link to another computer (e.g. via a telephone line), but can in practice be any device having a suitable driver and mnemonic (see section 28 of this document - RXnnnn - and Appendix 4 of 2.465(82)). The normal COPY command in PIP cannot be used because only 500 bytes or so of data are buffered in RAM before a disk access occurs, and during the disk access incoming data will be lost (unless the data rate is very low or deliberate pauses are inserted after carriage returns). Also, PIP is not suitable for binary data, i.e. data in which NUL (ASCII 00) or ETX (ASCII 03) are significant.

RDCAS reads the incoming data into RAM, at a maximum rate of 4800 baud, and after the data have been received writes them out to disk. The size of the RAM buffer is approximately 27 Kbytes; larger amounts of data can be handled but only if the sending device can be "rewound" to the beginning. In this case RDCAS writes the first buffer's worth to disk, prompts for the sending device to be rewound and then skips exactly the number of bytes already stored before starting to fill the buffer again. This can only work correctly if the device is "rewound" to precisely the same point as when the data were first received.

22.2 Preliminaries

If necessary, first load the required device driver (e.g. RX4800). Load RDCAS (normally at 0) at execute it at its load address.

22.3 User Input

Reply to the prompts as follows:

From ? Enter the mnemonic of the input device to be read, followed by <RETURN>.
<ESC> will exit to the monitor.

Filename ? Enter a disk filename (or other output device mnemonic), followed by <RETURN>. The device defaults to DK0: and the extension defaults to .S.

<ESC> will exit to the monitor.

After the prompt "Rewind and start tape; press any key when replaying leader:" RDCAS awaits a keypress before reading data from the input device. The input should be in an "idle" state (no characters being received) at this point.
Once reading has begun, pressing any key will cause the data read into the buffer to be written to disk. RDCAS will then exit to the monitor.

22.4 User Output

While the data are being read, they are displayed on the screen. This is principally to give confidence that the data are being received correctly. Control characters are displayed rather than being acted upon, so the format on the screen is not relevant.

If the RAM buffer fills, the contents are written to disk and the message "Rewind and start tape; press any key when replaying leader:" will again be displayed. Once the input device has been "rewound" and a key has been pressed reading will commence, but data will only be stored in the buffer after those characters already written to disk have been skipped.

22.5 Example

```
.L 0 cr
LOAD FROM? DK1:RDCAS cr
BEG ADDR 0000
EXECUTE  0000
END ADDR 014E
UNDEF SYM 00
#E cr

          ---  Read Cassette Utility  ---

Buffer length = 6B03 (hex)
From ? TI: cr
Filename ? JUNK cr

Rewind and start tape; press any key when replaying leader:
```

The above will result in data being read from device TI: (set up by RXnnnn - see section 28), buffered in RAM and being written to the disk file JUNK.S.

23. BIOS32 - Customisation patch for CP/M 2.2

File name:	BIOS32.O
Description:	Interfaces the CP/M Operating System to the EP1M/27.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Executable machine code utility.
Load method:	Monitor L command.
Load address:	3FF2; non-relocatable. BIOS32 copies itself to a higher address when executed.
Execute address:	3FF2 (hex).
Space occupied:	BIOS32 itself occupies 984 bytes, but once CP/M is loaded and running the entire 32 Kbytes of main RAM are utilised.

23.1 Description

CP/M 80 (registered trademark of Digital Research) is an "industry standard" operating system for the 8080/8085/Z80 range of microprocessors. BIOS32 allows this operating system to be run on the EP1M/27 (Zelda) thereby giving access to a very wide range of commercial disk-based software. CP/M can be purchased in an "unconfigured" form, suitable for a wide range of computers, which requires "patching" into the particular hardware environment of the machine; BIOS32 performs this patching.

The standard EP1M/27 allows only a "32K" CP/M system to be implemented. By adding an additional RAM unit (UN27/13) "61K" CP/M can be run. More details can be obtained from Monitoring and Control Section, Designs Department.

23.2 Preliminaries

Before BIOS32 can be used a suitable copy of CP/M must be purchased. What is required is "32K CP/M 2.2" on an 8" single-sided, single-density disk. If difficulty is encountered in buying a 32K version, a BIOS configured for a different memory size (less than 32K, e.g. 20K) can be provided.

Load BIOS32 from the utilities disk, insert the 32K CP/M disk into drive 0 and execute BIOS32 at its load address (3FF2).

23.3 User Input

None. If CP/M loads correctly control is transferred to its Console Command Processor. You are referred to one of the many books on CP/M for more details. If CP/M fails to load properly the prompt "Press any key to retry" will be displayed.

To return to normal Zelda operation a full reset is required (CTRL/BREAK).

23.4 User Output

If CP/M loads correctly the message "32K CP/M 2.2" is displayed and the Console Command Processor is entered ("A>" prompt). If CP/M fails to load correctly one of the following error messages will be produced:

Not a 32K CP/M system disk!
The disk in drive 0 does not contain a valid copy of CP/M.

Disk not ready at track n, sector m
The disk drive gate is not closed, no disk is inserted, or the disk is hard sectored. The track and sector numbers are given in decimal.

Track/sector not found at track n, sector m
The disk is either damaged or is the wrong format (e.g. double density).

CRC error at track n, sector m
A data or i.d. cyclic redundancy check error occurred.

Lost data at track n, sector m
The disk is the wrong format (e.g. 256 byte sectors).

Once the error message has been produced, pressing any key will cause the operation to be repeated.

23.5 Example

```
.L 0 cr
LOAD FROM? DK1:BIOS32 cr
BEG ADDR 3FF2
EXECUTE  3FF2
END ADDR 43C9
UNDEF SYM 00
#E cr

32K CP/M 2.2
A>
```

The system is now ready to receive CP/M commands.

24. SPOOL - Send console output to disk file

File name:	SPOOL.C
Description:	Opens a disk file for writing, to which subsequent console output is vectored.
File format:	"C-format" binary file as per 2.465(82) section 4.7.6.
File type:	Executable machine code utility.
Load method:	Monitor X command.
Load address:	FE30; non-relocatable.
Execute address:	FE30 (hex).
Space occupied:	159 contiguous bytes.

24.1 Description

SPOOL opens a file on the "source output" channel (channel 5) and vectors all subsequent console output to this file, as well as displaying it on the screen as normal. The spool file is closed, and spooling terminated, when the monitor is entered at its main re-entry point RENTRY (see 2.467(82) section 4.1). This will be on exit from a standard utility such as the assembler, on pressing <ESC> when in PIP or the editor, or on pressing "." (full point) when in the monitor.

SPOOL is useful if a permanent record is required of something normally written only to the screen, e.g. the differences found by the file comparator CMPARE (see section 4).

Because SPOOL uses the source output channel, care must be taken not to attempt to open a file on this channel while spooling is in progress. For example, the assembler must not be run with the listing or symbol table enabled and the editor W command must not be used. Care should also be taken not to change the disk while spooling is in progress.

24.2 Preliminaries

Execute SPOOL using the monitor's X command.

24.3 User Input

Reply to the prompt as follows:

SPOOL TO FILE: Enter the filename of the spool file, followed by <RETURN>. <ESC> will exit to the monitor without opening a file. The device defaults to DK0: and the extension to .S. A device other than disk may be specified.

24.4 User Output

The message "SPOOLING...." indicates that the spool file has been opened and that subsequent console output will be written to this file as well as to the screen. When the monitor is entered at RENTRY the message "....SPOOLING TERMINATED" is produced and the file is closed.

24.5 Example

```
.X cr
EXECUTE BINARY FILE: DK1:SPOOL cr
SPOOL TO FILE: S cr
SPOOLING....
.cr cr
*L cr
DK0:ZELHB3.1   696 SECTORS
DK0:IDBM  .S    35 SECTORS
DK0:ZELHB3.2   283 SECTORS
DK0:GE1597.T   409 SECTORS
DK0:S     .S    20 SECTORS
    Unused     520 SECTORS
*esc
....SPOOLING TERMINATED
.
```

The above example illustrates spooling to disk the disk directory listing produced by PIP's LIST command. This might be done so that it could be printed out later, or perhaps to produce the input file for an alphabetical sort program.

25. SUBMIT - Get console input from disk file

File name:	SUBMIT.C
Description:	Opens a disk file and vectors the console input to be taken from this file.
File format:	"C-format" binary file as per 2.465(82) section 4.7.6.
File type:	Executable machine code utility.
Load method:	Monitor X command.
Load address:	FE30; non-relocatable.
Execute address:	FE30 (hex).
Space occupied:	86 contiguous bytes.

25.1 Description

SUBMIT opens a file on the "source input" channel (channel 4) and vectors the console input to be taken from this file until ETX (ASCII 03) is read, whereupon the console input is restored to normal (keyboard).

SUBMIT allows the use of a "command file" which is executed exactly as if it had been typed at the keyboard. Such a command file might, for example, contain a number of operations which need to be executed repetitively: such as the commands required for the linking loader to link several object modules together. The command file is most easily created using VIDIT in binary mode (see section 16) as a precise sequence of control characters (carriage return, escape etc.) is often vital. Line feeds are generally not required.

Because SUBMIT uses the source input channel, care must be taken to ensure that none of the commands in the file require a file to be opened on this channel. For example, the assembler cannot be run from a command file as it uses the source input channel itself.

SUBMIT and SPOOL cannot co-exist.

25.2 Preliminaries

Execute SUBMIT using the monitor's X command.

25.3 User Input

Reply to the prompt as follows:

SUBMIT: Enter the filename of the command file, followed by <RETURN>. <ESC> will exit to the monitor without opening a file. The device defaults to DK0: and the extension to .S.

25.4 User Output

None. If an error occurs while the command file is being processed, console input will continue to be taken from the file until the ETX is encountered.

25.5 Example

```
.X cr
EXECUTE BINARY FILE: DK1:SUBMIT cr
SUBMIT: S cr
.L 0,5000
LOAD FROM? MAIN,EXEC,EVAL,LINK,PATCH
BEG ADDR 0000
EXECUTE  0000
END ADDR 0BFF
UNDEF SYM 53
BEG ADDR 0C00
END ADDR 1C40
UNDEF SYM 70
BEG ADDR 1C41
END ADDR 2E64
UNDEF SYM 54
BEG ADDR 2E65
END ADDR 2EE0
UNDEF SYM 51
BEG ADDR 2F00
END ADDR 31FF
UNDEF SYM 00
#.
.
```

In the above example, the command file contained the following:

```
L0,5000cr
MAIN,EXEC,EVAL,LINK,PATCHcr
cr
cr
cr
cr
cr
.
```

26. MON - Display memory contents on hex displays

File name:	MON.C
Description:	Sets up an interrupt-driven background task which updates the hex displays on the UN26/31.
File format:	"C-format" binary file as per 2.465(82) section 4.7.6.
File type:	Executable machine code utility.
Load method:	Monitor X command.
Load address:	FECB; non-relocatable.
Execute address:	FEDF (hex).
Space occupied:	53 contiguous bytes.

26.1 Description

MON sets up a periodic interrupt by the Z80-CTC on the CO4/6 unit. This interrupt takes place at 30 Hz and updates the hexadecimal displays on the UN26/31 to the memory contents addressed by the hexadecimal switches. The right-hand pair of digits display the data at the selected address and the left-hand pair the data at the next address. The interrupts will continue until the system is next reset (CTRL/BREAK) or until a program is run which disables the CTC interrupt.

Most standard system utilities (assembler, editor etc.) are compatible with the interrupts and can be run whilst memory locations are monitored. Interrupt mode 2 is used, so care must be taken not to alter the contents of the I register or the interrupt mode itself. In particular the monitor's copy of the user's I register (:I) must not be altered as this is copied into I whenever an E or S monitor command is issued. If interrupts are disabled (By changing :IF or directly by the Z80's DI instruction) the memory monitoring will be suspended, but will restart when interrupts are enabled again. Permanent cessation of interrupts can only be achieved by a full reset or by writing to the Z80-CTC.

The interrupt service routine resides at the top of memory page FE so utilities which use this area (e.g. SPOOL) must not be loaded while MON is active.

26.2 Preliminaries

Execute MON using the monitor's X command.

26.3 User Input

None. The four hexadecimal thumbwheel switches on the front-panel of the UN26/31 are used to set up the address to be monitored.

26.4 User Output

None. The right-hand pair of hexadecimal displays indicate the data at the selected address, and the left-hand pair the data at the next address (i.e. address+1)

26.5 Example

```
.X cr
EXECUTE BINARY FILE: DK1:MON cr
.
```

27. DISFUN - 256 character display driver

File name:	DISFUN.O
Description:	VDU driver providing a different symbol for each of the 256 possible character values.
File format:	"Intel" object code as per 2.465(82) Appendix 3.
File type:	Machine code output driver.
Load method:	Monitor L command.
Load address:	FE00; non-relocatable.
Execute address:	None.
Space occupied:	59 contiguous bytes plus four bytes in the RAM mnemonic table at FF4F (hex).
Driver mnemonic:	DF:

27.1 Description

DISFUN is a VDU driver which displays symbols for all characters including the ASCII control characters. The displayed symbol also indicates if the character received had bit 7 set or reset. The only character acted upon by DISFUN is carriage return (0DH) which results in a new line being performed as well as the carriage return symbol being displayed. All other control codes result in a displayed symbol only. DISFUN allows the contents of a binary file to be displayed, and also makes it possible to examine for embedded control characters in a file.

The symbols displayed for each character value are shown in Appendix 1. If the character is in the range 0 to 7FH the symbol is displayed in normal video (a green symbol on a black background). Characters in the range 80H to FFH are displayed as the inverse video symbol for the corresponding character with bit 7 reset.

The mnemonic DF: is assigned to the calling address of DISFUN, and this is placed in the RAM mnemonic table when the utility is loaded. The mnemonic is placed at FF4FH, overwriting BB:, and any mnemonics beyond this address are removed from the table.

27.2 Preliminaries

Load DISFUN using the monitor L command but do not execute it. Note that the relocating, linking loader cannot be used.

27.3 User Input

DISFUN may be assigned to an output channel in the usual way, using the mnemonic DF:. This may be done in response to prompts for filenames or via the monitor M command, for example.

27.4 User Output

Characters sent to DISFUN are displayed as shown in Appendix 1. Characters from 0 to 7FH are displayed in normal video, and characters 80H to FFH in inverse video.

27.5 Example

```
.E :PI cr
*C DF:=TEST cr
Text file line 1 CR
LF Text file line 2 CR
LF Last line of text file CR
LF EX
*
```

28. RXnnnn - RS232 receive drivers

File names:	RX110.C, RX300.C, RX1200.C, RX2400.C, RX4800.C, RX9600.C
Description:	Input drivers for the auxiliary RS232 port (SKC).
File format:	"C-format" binary files as per 2.465(82) section 4.7.6.
File type:	Machine code input driver.
Load method:	Monitor X command.
Load address:	FC80; non-relocatable.
Execute address:	FC80 (hex).
Space occupied:	103 contiguous bytes.
Driver mnemonic:	TI:

28.1 Description

The driver programs RX110, RX300, RX1200, RX2400, RX4800 and RX9600 provide access to the auxiliary RS232 input at socket SKC on the EP1M/27 (see DDHB 2.464(82) section 12.3). The appropriate baud rate and UAR/T parameters are set up when the driver is executed: the UAR/T is set to 8 data bits, no parity. Note that the receive and transmit channels have a common clock so executing TXnnnn or TOnnnn may affect the receive baud rate. These drivers do not support hardware handshaking.

The mnemonic TI: is assigned to the calling address of RXnnnn, and this is automatically added to the RAM mnemonic table when the utility is executed. The mnemonic is added to the end of the table and existing mnemonics are not affected. If the mnemonic TI: is found to be present already, it is left unchanged.

"Immediate return" mode is implemented (see DDHB 2.465(82) Appendix 4).

28.2 Preliminaries

Execute the appropriate driver using the monitor's X command.

28.3 User Input

RXnnnn may be assigned to an input channel in the usual way, using the mnemonic TI: in response to a prompt for an input device/file. In particular, TI: will probably be specified as the input device for RDCAS (see section 22). Because handshaking is not implemented, the utility taking input from TI: must be able to accept data at the incoming rate without bytes being lost.

28.4 User Output

None.

28.5 Example

```
.X cr
EXECUTE BINARY FILE: DK1:RX300 cr
.
.L 0 cr
LOAD FROM? DK1:RDCAS cr
BEG ADDR 0000
EXECUTE  0000
END ADDR 014E
UNDEF SYM 00
#E cr

          ---  Read Cassette Utility  ---

Buffer length = 6B03 (hex)
From ? TI: cr
etc...
```

29. TXnnnn - RS232 transmit drivers

File names:	TX110.C, TX300.C, TX1200.C, TX2400.C, TX4800.C, TX9600.C
Description:	Output drivers for the auxiliary RS232 port (SKC).
File format:	"C-format" binary files as per 2.465(82) section 4.7.6.
File type:	Machine code output driver.
Load method:	Monitor X command.
Load address:	FC00; non-relocatable.
Execute address:	FC00 (hex).
Space occupied:	103 contiguous bytes.
Driver mnemonic:	TO:

29.1 Description

The driver programs TX110, TX300, TX1200, TX2400, TX4800 and TX9600 provide access to the auxiliary RS232 output at socket SKC on the EP1M/27 (see DDHB 2.464(82) section 12.3). The appropriate baud rate and UAR/T parameters are set up when the driver is executed: the UAR/T is set to 8 data bits, 1 stop bit, no parity. Note that the receive and transmit channels have a common clock so executing RXnnnn may affect the transmit baud rate. These drivers support hardware handshaking to the extent that the UAR/T will not transmit data unless the CTS (Clear to Send) signal is active, i.e. at a positive level at pin 20 of SKC.

The mnemonic TO: is assigned to the calling address of TXnnnn, and this is automatically added to the RAM mnemonic table when the utility is executed. The mnemonic is added to the end of the table and existing mnemonics are not affected. If the mnemonic TO: is found to be present already, it is left unchanged.

"Immediate return" mode is implemented (see DDHB 2.465(82) Appendix 4).

29.2 Preliminaries

Execute the appropriate driver using the monitor's X command.

29.3 User Input

TXnnnn may be assigned to an output channel in the usual way, using the mnemonic TO: in response to a prompt for an output device/file. In particular, TO: may be specified as the output device for MFT (see section 5).

29.4 User Output

None.

29.5 Example

```
.X cr
EXECUTE BINARY FILE: DK1:TX300 cr
.
.L 0 cr
LOAD FROM? DK1:MFT cr
BEG ADDR 0000
EXECUTE  0000
END ADDR 02FF
UNDEF SYM 00
#E cr

Multi-File Transfer Utility V1.0
List files: WOMBAT cr
Destination? TO: cr
etc...
```

30. TOnnnn - RS232 transmit drivers with inter-file pause

File names: TO300.C, TO1200.C, TO2400.C, TO4800.C
Description: Output drivers for the auxiliary RS232 port (SKC).
File format: "C-format" binary files as per 2.465(82) section 4.7.6.
File type: Machine code output driver.
Load method: Monitor X command.
Load address: FC00; non-relocatable.
Execute address: FC00 (hex).
Space occupied: 112 contiguous bytes.
Driver mnemonic: TO:

30.1 Description

The driver programs TO300, TO1200, TO2400, and TO4800 provide access to the auxiliary RS232 output at socket SKC on the EPIM/27 (see DDHB 2.464(82) section 12.3). The appropriate baud rate and UAR/T parameters are set up when the driver is executed: the UAR/T is set to 8 data bits, 1 stop bit, no parity. Note that the receive and transmit channels have a common clock so executing RXnnnn may affect the transmit baud rate. These drivers support hardware handshaking to the extent that the UAR/T will not transmit data unless the CTS (Clear to Send) signal is active, i.e. at a positive level at pin 20 of SKC. If bit 3 of E (initialise) is set on input to the driver a delay of approximately 10 seconds results, this is intended to provide an inter-file gap when used to write multiple files to cassette tape (via a suitable modem) with MFT.

The mnemonic TO: is assigned to the calling address of TOnnnn, and this is automatically added to the RAM mnemonic table when the utility is executed. The mnemonic is added to the end of the table and existing mnemonics are not affected. If the mnemonic TO: is found to be present already, it is left unchanged.

"Immediate return" mode is implemented (see DDHB 2.465(82) Appendix 4).

30.2 Preliminaries

Execute the appropriate driver using the monitor's X command.

30.3 User Input

TOnnnn may be assigned to an output channel in the usual way, using the mnemonic TO: in response to a prompt for an output device/file. In particular, TO: may be specified as the output device for MFT (see section 5).

30.4 User Output

None.

30.5 Example

```
.X cr
EXECUTE BINARY FILE: DK1:TO300 cr
.
.L 0 cr
LOAD FROM? DK1:MFT cr
BEG ADDR 0000
EXECUTE  0000
END ADDR 02FF
UNDEF SYM 00
#E cr

Multi-File Transfer Utility V1.0
List files: .I cr
Destination? TO: cr
etc...
```

The above example illustrates sending all .I files to a cassette tape, with a ten second gap between files (and before the first file).

Documentation has not yet been prepared for sections 31 to 50 of this handbook. It is anticipated that Issue 2 of DDHB 2.466(82), containing these sections, will be published during 1984. Meanwhile, information can be obtained on request from the authors at Monitoring and Control Section, Designs Department.

A P P E N D I X 1

DISFUN Character Set

D56070 A4

D56070A4

ORIGINAL FRAME SIZE 190mm x 277mm

THIRD ANGLE PROJECTION

All dimensions in millimetres unless otherwise stated :
Normal tolerances :
no decimal place — ±1 mm unless
one decimal place — ±0·3 mm otherwise
two decimal places — ±0·1 mm stated

DISFUN Character Set

Data bits 0-3	Data bits 4-7							
	0	1	2	3	4	5	6	7
0	$^{N}_{U}$	—		Ø	@	P	‘	p
1	$^{S}_{H}$	⌐	!	1	A	Q	a	q
2	$^{S}_{X}$	├	"	2	B	R	b	r
3	$^{E}_{X}$	┼	#	3	C	S	c	s
4	$^{E}_{T}$	┬	$	4	D	T	d	t
5	$^{E}_{Q}$	┴	%	5	E	U	e	u
6	$^{A}_{K}$	⌋	&	6	F	V	f	v
7	𝔅	┐	’	7	G	W	g	w
8	$^{B}_{S}$	┘	(	8	H	X	h	x
9	$^{H}_{T}$	┤	)	9	I	Y	i	y
A	$^{L}_{F}$	└	*	:	J	Z	j	z
B	$^{V}_{T}$	$^{E}_{C}$	+	;	K	[	k	{
C	$^{F}_{F}$	$^{F}_{S}$	,	<	L	\	l	¦
D	$^{C}_{R}$	$^{G}_{S}$	–	=	M	]	m	}
E	$^{S}_{O}$	$^{R}_{S}$	.	>	N	↑	n	~
F	$^{S}_{I}$	$^{U}_{S}$	/	?	O	_	o	▨

Symbols displayed for data values 0-7F.

Note:-
Characters 00-7F are green on black background.
Characters 80-FF are as for 0-7F, but black on green background.

SCALE 0

BBC

DS/A4

CHANGE 4·11·83

ISS 1

DISFUN CHARACTER SET

DRN. P.P.
TCD.
CKD.
APPD

DESIGNS DEPARTMENT

D56070A4