Programmers' Guide

TORCH
COMPUTERS

Programmers' Guide

TORCH
COMPUTERS

TORCH COMPUTERS LIMITED
Abberley House, Great Shelford, Cambridge CB2 5LQ

#1504, 50 Milk Street, Boston, Mass. 02109 USA

#212, 7240 Woodbine Avenue, Markham,
Ontario, CANADA

Preliminary issue

TORCH PROGRAMMERS' GUIDE: SECTION 0

CONTENTS

TORCH PROGRAMMERS' GUIDE: SECTION 1

INTRODUCTION

1.0 System Overview and Guide Contents

This guide outlines the facilities available to the programmer on the Torch Computer, especially the handling of peripherals.

The Torch Computer consists of a Z80 processor coupled to the input/output ports of the Torch Base peripheral processor. Programs are run on the Z80 card, and all commands for peripheral handling are sent to the Torch Base processor from the Z80. Similarly, input is passed from the Torch Base processor to the Z80. Facilities are provided for interfacing this communication at a high level (i.e. from the Z80, using commands built in at the same level as the program that is running) by the CPN operating system. For the convenience of the user, this is fully compatible with the CP/M operating system, allowing the majority of Z80 business software to be run on the Torch with no modification.

Full details of CPN, and its interfacing, are provided in section 2. This information will be of most interest to systems programmers, and compiler writers.. It is normally interfaced using Z80 assembler, although some languages will also allow it to be accessed. If you are programming using high level languages, you need not concern yourself with Section 2, since all interfacing to the system is already doe by the language using CPN.

Extensive facilities for the use of graphics, and for a multi page, multi window screen display (facilitating the implementation of a full word processor, and menu display) are provided by Torch VDU Control Codes. These are a high level interface, and may easily be called from any programming language supported by the Torch. They are detailed in section 3, which should be of interest to all programmers.

The Torch Base peripheral processor is based on the main 6502 p.c.b. of an Acorn B.B.C. microcomputer, and it handles the following peripherals:

a keyboard,

a colour/monochrome high resolution monitor,

two 5 1/4 inch double sided floppy discs (80 tracks with 10 sectors/side of each track),

a printer port,

a bidirectional RS 423 port,

a three channel analogue to digital channel,

a bidirectional Econet port,

and a bidirectional Prestel port.

Section 4 (Use of the Torch Base Processor) outlines the use of these facilities at a low level of interfacing. It should be of interest to systems programmers only; most of the features are accessed using Z80 assembler. Wherever possible, higher levels of interfacing should be used (i.e. the CPN operating system, and Torch VDU control codes; see sections 2 and 3), since Torch Computers Ltd. reserve the right to change the Torch Base Processor to upgrade the machine.

Section 5 contains some miscellaneous information likely to be of use to all programmers. This includes details of additional CCCP commands, and of keyboard output codes.

There are, of course, a large number of languages available for use on the Torch Computer. These include: Ada (TM), Algol 60, APL, Basic, BCPL, C, Cobol, Comal, Cross assemblers, FPL, Forth, Fortran, Imp 77, Lisp, Modula, Pascal, Pilot, PLM, PL/1, Prolog, Ratfor, Stage 2, and Z80 assembler. Of these, BCPL, C, Pascal, and Z80 assembler are languages that you are most likely to find useful.. It is not the place of this manual to detail the individual languages; books and manuals doing so are available from your dealer.

TORCH PROGRAMMERS' GUIDE: SECTION 2

CPN 2.2 INTERFACE INFORMATION

2.0 CONTENTS

2.1 INTRODUCTION

This section contains information for the programmer who wishes to write programs to operate under the Torch CPN operating system. It contains details of memory and system organisation, and system entry points. It also contains information needed to use the peripheral and disc I/O facilities of the Torch Computer. It is most likely to be of interest to systems programmers, and compiler writers.

Where a programmer has a choice between using CPN and using Torch Base commands, CPN commands are preferable; indeed, in general Torch Base commands should not be used.

There are four parts to memory when running under the CPN operating system:

i) The Torch Control Kernel (TCK)

ii) The Basic Disc Operating System (BDOS)

iii) The Cambridge Central Command Processor (CCCP)

iv) The Program Load Area (PLA)

The TCK module resides on the the 6502 PCB, and controls many of the I/O functions. Its interface is effectively used as a single module with BDOS, with a common entry address. Together, they are referred to as the Torch Disc Operating System (TDOS). TDOS is used by the CCCP module to give the user easy access to all disc held information. The PLA is the area of memory used to execute non-resident operating system command modules and user programs.

There is a small area of memory (0000 hex to 0100 hex) below the PLA which is reserved for systems information, and is detailed in section 2.2.1 (Low Memory)

2.2 MEMORY LAYOUT

2.2.0 Memory Map

```
---------------------------------------------------------------------
           |                                        |        ^
           |      TDOS (BDOS + TCK)                 |        |
TORBASE    |----------------------------------------|      Upper
           |                                        |      Memory
           |      CCCP                              |
COMBASE    |----------------------------------------|
           |                                        |
           |                                        |
           |      PLA                               |
           |                                        |
0100 hex   |----------------------------------------|
           |                                        |
           |      System parameters                 |
0000 hex   |----------------------------------------|
```

The exact memory addresses of COMBASE and TORBASE will vary, depending on the CPN version being used, but the systems parameters always run at the base of random access memory, from 0000 hex to 0100 hex. They contain the code (starting at 0000 hex) used to perform a warm start on the system. Each warm start loads and initialises all CPN and CCCP code before returning control to the CCCP. This means that to return to control of the CPN commands, all that any program has to do is jump to location 0000 hex, at which point the system will be reinitialised. In this case, memory from 0100 hex up to TORBASE - 1 is available for programs to run in.

2.2.1 Low Memory

There are details given below of the locations of the systems parameters in low memory:

0000	JP BIOS + 3
0003	0
0004	Default drive number (initially zero)
0005	JP TDOS
0008-2F	Reserved for future expansion
0030	JP ShortBdosEntry
0033-34	Logged in drive vector (initially 0001)
0035-36	Read/Only vector (initially 0)
0037	Reserved for future CPN use
0038-3A	Reserved for Debugger vector
003B-3C	DMA address
003D-3E	User stock save area
003F	Reserved for future CPN use
0040-4F	CBIOS scratchpad
0050-5B	Reserved for future CPN use
005C-7C	Default FCB
007D-7F	Default FCB random record count
0080-FF	Default DMA buffer
0100	Start of Program Load Area

2.2.2 TDOS Entry

Entry to TDOS is made at location 0005 hex where a jump instruction to TORBASE is found. (It follows that the address of TORBASE may be found at 0006 hex. If the CCCP is to be overlayed by user or operating system programs, it is therefore possible to calculate the size of available user memory.)

2.2.3 Running Programs in the PLA

Once loaded into memory, programs are run in the following manner:

The operator enters a command, optionally followed by a string of characters. Where a command requires the passing of filenames, the first string(s) of characters found that could be filenames are used. As a convenience, the Dynamic Memory Address buffer (DMA buffer; 0080 to 00FF hex by default; see section 2.4) is set to the tail end of the command line (that part of the line following the program name of the command). The first position of the buffer is set to the number of characters, excluding the final (carriage return). It is followed by the characters typed, with lower case letters translated to upper case.Lastly, there is a carriage return, followed by uninitialised memory. Thus if the following were typed:

```
COPY DAHA to juss
```

the DMA would be set to:

```
80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8Fhex
13  ' ' 'D' 'A' 'H' 'A' ' ' 'T' 'O' ' ' 'J' 'U' 'S' 'S' (cr)?????
```

For those commands followed by filenames, a File Control Block (FCB) will be set up by the CCCP in the area reserved for default FCB use (005C - 007F hex). These are used for accessing the files via TDOS. More detail is contained later in this manual, in section 2.4 (File Control Blocks).

The command may be one of the standard CPN commands, in which case it is executed immediately; or it may be a user command (which will also be the name of a user program). If it is the latter, the file (command).COM will be loaded from disc into the PLA starting at 0100 hex.

The CCCP gives control to the user program, which will then be executed. Since the user program was entered from the CCCP, control may be simply returned to it upon completion, using the Z80 code 'RET'; in this case memory above the PLA cannot be used. If required, control can be resumed at the CPN command level by the use of a jump to 0000 hex; in this case, memory up to TORBASE - 1 is available, since the system will be reloaded and reinitialised.

2.3 FILE STRUCTURE

2.3.0 File Names

In CPN, a file is referred to by a disc file name. It has three parts: one drive select letter, a file name consisting of between one and eight non-blank characters, and a file type consisting of between zero and three non-blank characters. If no drive select letter is used, then the current default disc is used; initially this is the logged on drive, but it may be changed using CPN function 14 (Select Disc). The file name is used to distinguish between different files of the same type. The file type indicates the nature of the file, using a locally agreed convention; some of the more commonly used conventions are given below.

MAC	Macro Assembler Source	BPL	BCPL Source File
PRN	Printer Listing	PAS	Pascal Source File
COM	CCCP Command File	BAS	Basic Source File
HEX	Hex Machine Code	C	C Source File
REL	Relocatable Module	BAK	Backup File
SUB	Submit File	$$$	Temporary File

2.3.1 Disc Layout

Each disc contains a directory of the files on it, as well as an area of file data. This format allows a disc to have a variable number of records on it, and a file to be spread out over parts of the data area of a disc that are not physically contiguous.

The file itself is stored on the data area of the disc in up to 64K records (numbered from 0 to 65535) of 128 bytes, giving a maximum length of 8 Mbytes. A group of 128 records (16 Kbytes) is known as an extent, and is a measure used when accessing files sequentially. As will be shown in the next section, a maximum of 256 extents may be accessed on any one file, and so it is only possible to access 4 Mbytes sequentially. If larger files are to be used, then they must be randomly accessed.

2.4 FILE CONTROL BLOCKS

All disc I/O is handled by file control blocks, which consist of a representation of the disc file name being used, and some system information. Most CPN functions from 15 upwards use the register pair DE to address an FCB. An FCB is 33 bytes long for sequential access of files, and 36 bytes long for random access; there is a default FCB area reserved in memory by CPN at 005C hex, of length 36 bytes. The memory immediately above this, from 0080 to 00FF hex (128 bytes, or 1 record in length) is the default DMA, which is the region of memory normally used for passing records to and from disc.

An FCB has the following structure:

```
 00 01 02 .... 08 09 10 11 12 13 14 15 16 17 .... 31 32 33 34 35
-----------------------------------------------------------------
|dr|f1|f2|/  /|f8|t1|t2|t3|ex|h1|h2|rc|u0|u1|/  /|un|cr|r0|r1|r2|
-----------------------------------------------------------------
```

dr Drive code (0-16)

0 implies that the current default drive is used.

Otherwise, the numbers 1 to 16 refer to drives A to P respectively.

See Function 14 (Select Disc) for details of selection of the default drive.

f1-f8 Contains the file name in upper case ASCII code, with the high bit set to 0.

See Function 30 (Set File Attributes) for details of use of the high bits in user programs.

t1-t3 Contains the file type in upper case ASCII code. The high bits of t1, t2 and t3 are used as follows:

t1: set to 1 for Read/Only file, otherwise to 0.

t2: set to 1 for a SYS file (no directory listing), otherwise to 0.

t3: set to 1 for a file that is open, and in the same state as when opened; otherwise to 0.

ex Contains the current extent number. This covers the range 0 - 255 during file I/O, but is normally set to 0 by the user.

For more information, see section 2.3.1 (Disc Layout).

h1 Used internally by the system.

h2 Used internally by the system. This byte is set to zero by the system on a call to OPEN, MAKE or SEARCH functions.

rc Contains a record count for 'ex', the current extent number. The count is in the range 0 - 128.

u0-un Contains the user field. This is normally unused by CPN, but see below for its use in the default FCB area, with calls having two parameters.

It is also used in some CPN functions, as detailed below.

cr Contains the current record of a file to use in a sequential I/O operation. When reading all of a file sequentially, it is set to 0 by the programmer, since the system automatically increments this value on each sequential read.

r0-r2 Contains the current record to use in a random I/O operation. The value is contained in registers r0 and r1, with r0 the low order byte. Overflow goes to r2.

If a filename is passed with a command to CCCP, then an FCB is set up (with the structure given above) in the default FCB region (005C - 007F hex). If the command has two filenames, then the second filename is made into an FCB in the user field of the default FCB. Note that it is the responsibility of the programmer to clear the cr and ex bytes of the user field before opening the file.

2.5 OPERATING SYSTEM CALLS

2.5.0 Accessing CPN Functions

To communicate with the keyboard, the disc operating system, and other external peripherals, the user program will use CPN I/O facilities. To access the I/O system a function number must be passed to CPN, commonly with some other passed values; e.g. to delete a file, CPN must be passed function number 19 (Delete File), and the address of an FCB which is used to identify the file to delete. CPN often gives a returned value; in this case indicating success or failure.

There are two ways of passing these values to CPN. The first is by a jump to location 0005 hex. The function number is passed in register C, and any other values are passed in register E, or the double register DE. Results are returned in register A if of single byte length, with L = A; or in registers HL if of double byte length, with A = L and B = H for historical reasons. Thus with single byte results, only registers A and L are affected; but with double byte results A, B, C, H and L may be changed.

The second method is a call to location 0030 hex; the byte following the call must be the function number. Thus the call may be performed by the single byte instruction:

```
        RST  0030H                      ;followed by
        DB   (function number)
```

Values are passed to the call in register E, or in the double register DE; results are returned in register A, or in registers HL (with register A set to the value in L). Thus with single byte results only register A is altered, but with double byte results registers A, H and L are affected.

The differences between the above calls should be noted. The first method is three byes longer in instructions. With single byte results, it affects register L, and with double byte results, it affects registers B and C; whereas the second method does not. In general the second method is to be preferred, unless software is being written to be compatible with operating systems other than CPN.

2.5.1 Function List

Available CPN functions are given in a list below, and then outlined in detail in the next section. Those functions marked with an asterisk are provided for compatibility only.

0		System Reset
1		Keyboard Input
2		Screen Output
3		Raw Keyboard Input
4		Raw Screen Output
5		Printer Output
6		Direct Console I/O
7	*	Get I/O Byte
8	*	Set I/O Byte
9		Display String
10		Read Keyboard Buffer
11		Get Keyboard Status
12		Return Version Number
13		Reset Disc System
14		Select Disc
15		Open File
16		Close File
17		Search For First
18		Search For Next
19		Delete File
20		Read Sequential
21		Write Sequential
22		Make File
23		Rename File
24		Return Login Vector
25		Return Current Disc
26		Set DMA Address
27	*	Get Address (Alloc)
28		Write Protect Disc
29		Get Read/Only Vector
30		Set File Attributes
31		Get Address (Disc Parms)
32		Set/Get User Code
33		Read Random
34		Write Random
35		Compute File Size
36		Set Random Record
37		Reset Drive
40		Write Random With Zero Fill

2.5.2 Call Specifications

Function 0: System Reset

Passed Values	Returned Values
None	None

This function re-enters the CPN operating system via the CCCP module. It has exactly the same effect as jumping to 0000 hex, namely, the disc system is reinitialised.

Function 1:Keyboard Input

Passed Values	Returned Values
None	Register A: ASCII character

This function reads a character from the keyboard to register A. If no character has been typed, execution is suspended until one is typed, and it is then read.

Characters are reflected to the screen, after treatment as follows:

Graphics character	Reflected.
Carriage return	Reflected.
Line feed	Reflected.
New line	Reflected.
Tab	Expanded into a column of 8 spaces, then reflected.
Escape	Invokes Torch VDU controls; see Section 3.
Other control characters	Trapped and not reflected.

Function 2: Screen Output

Passed Values	Returned Values
Register E: ASCII character	None

This function reads a character (in ASCII code) from register E, and reflects it to the screen. Control characters are treated in the same way as in function 1 (Keyboard Input).

Function 3: Raw Keyboard Input

Passed Values	Returned Values
None	Register A: ASCII character

This function stores the next character typed at the keyboard in register A. There is no interpretation of any of the characters, including control characters. The function neither requires that the terminal be attached, nor does it attach it.

(Since the Torch does not support a paper tape reader, this function is not used for Reader Input.)

WARNING: Wherever possible, use of this function should be avoided, since it avoids all normal interpretation of control characters. It is available under CPN for specialist programming only.

Function 4: Raw Screen Output

Passed Values	Returned Values
Register E: ASCII character	None

This function outputs the character stored in register E to the screen. There is no interpretation of the output character; so, for instance, tabs are not expanded, and printer echo is not checked for.

See Section 4.8 (Character Output) for full details.

(Since the Torch does not support a paper tape punch, this function is not used for Punch Output.)

WARNING: Wherever possible, use of this function should be avoided, since it avoids all normal interpretation of control characters. It is available under CPN for specialist programming only.

Function 5: Printer Output

Passed Values	Returned Values
Register E: ASCII character	None

This function reads the character stored (in ASCII code) in register E, and outputs it to the printing device.

Function 6: Direct Console I/O

Passed Values	Returned Values
Register E: FF hex (input)	Register A: ASCII character or or status (input)
or ASCII character (output)	or No value (output)

This function provides facilities for unadorned console I/O. Input is selected by setting register E to FF hex on entry; the returned value in register A is either 00 (no character ready) or the next character to have been typed. Output is selected by passing any value other than FF hex in register E; the value is treated as ASCII code for a character, and is sent to the screen.

Section 4.8 (Character Output) provides a full description of direct console I/O on the Torch.

WARNING: Wherever possible, use of this function should be avoided, since it avoids all normal interpretation of control characters. It is available under CPN for specialist programming only.

Function 7: Get I/O Byte

Passed Values	Returned Values
None	Register A: I/O Byte value

This function returns the current value of IOBYTE in register A. It is provided for historical reasons only.

Function 8: Set I/O Byte

Passed Values	Returned Values
Register E: I/O Byte value	None

This function sets the value of IOBYTE to the value of register E. It is provided for historical reasons only.

Function 9: Display String

Passed Values	Returned Values
Registers DE: String address	None

This function reads a string, addressed by registers DE, and reflects it to the screen. The string is terminated by the character '$', which is not reflected to the screen. Characters are otherwise treated as in function 1 (Keyboard Input).

Function 10: Read Keyboard Buffer

Passed Values	Returned Values
Registers DE: Buffer address	Characters in keyboard buffer

This function reads a line of keyboard input into a buffer, addressed by registers DE.

The keyboard input may be edited as it is input, using the following codes:

Rubout/Delete/ Control-H	Remove last character to be entered on the line.
Control-C	Reboot (only when at start of line).
Control-E	Cause physical end of line.
Control-J (lf)/ Control-M (cr)	End line of input.
Control-R	Retype current line after '# (new line)'
Control-U	Remove current line after '# (new line)'
Control-X	Remove characters to start of line.

(Start of line is defined as the first character position after the prompt. No editing code may move back beyond this.)

Input is ended either by the input buffer overflowing, or by (newline) or (carriage return). The buffer takes the form:

DE	->	Buffer length, M (1 to 255 characters)
DE+1	->	Number of characters read, N
DE+2 to DE+1+M	->	Rest of buffer.

The 'rest of buffer' consists of the characters typed at the keyboard; if N < M, then all positions past the Nth character are uninitialised.

Function 11: Get Keyboard Status

Passed Values	Returned Values
None	Register A: Console status

This function checks whether a character has been typed at the keyboard. If a character is ready, FF hex is returned in register A; otherwise, 00 hex is returned.

Function 12: Return Version Number

Passed Values	Returned Values
None	Registers HL: Version number

This function returns a code in registers HL for the version of CPN implemented. H is set to 00 hex, and L returns a hex representation of the version, e.g.

CPN 2.5 => 25 hex, CPN 3.10 => 3A hex.

This function is mainly useful for implementation dependent software, e.g. producing an error message if an earlier operating system does not implement a desired utility.

Function 13: Reset Disc System

Passed Values	Returned Values
None	None

This function resets the state of all discs in the filing system to Read/Write (see functions 28,29), selects drive A and sets the default DMA (see function 26 and section 2.4) to 0080 hex.

Function 14: Select Disc

Passed Values	Returned Values
Register E: Disc to select	None

This function sets the default disc for the system, according to the code in register E. 00 hex represents drive A, 01 hex represents drive B, and so on up to a maximum of 0F hex for drive P on a full 16 user system, such as the Torch Net system.

The default drive is used whenever an FCB specifies a drive code of 0. It can be overridden by direct selection of drives A - P, using drive codes of 1 - 16.

Function 15: Open File

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function opens an already created file in the current user's disc directory. TDOS will scan the relevant directory for an FCB matching that addressed by registers DE in bytes 1-14; a '?' will match any character in the scanned directory. (This is a useful facility when using wild cards in the original command; if it is used then the first match made is used). The system automatically zeroes bytes ex, r1 and r2; if it is desired to access a file sequentially from the first record, then byte cr must be zeroed by the programmer.

If a match is made, then bytes 00-15 of the matched FCB in the directory are copied into the user field of the FCB, and register A is set to 00 hex. If no match could be made, then no alteration is made to the user field of the FCB, and register A is set to FF hex. In either case, the h2 byte of the FCB is cleared.

Note that any existing file must not be accessed before it has been opened.

Function 16: Close File

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function closes a file after it has been used: it is not needed if a file has only been read, but is necessary if a file has been written to. The FCB addressed by DE is matched in the same way as in function 15 (Open File).

If a match is made, then 00 hex is returned in register A; if in addition the file was originally Opened or Made, then the new FCB is permanently recorded in the referenced disc directory. If no match is made, then FF hex is returned in register A.

Function 17: Search For First

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function examines the file directory for the first occurence of an FCB matching that addressed by registers DE. The match is performed as in function 15 (Open File). If the dr byte is set to '?' then the auto disc select function is disabled, and the default disc is searched; the first match belonging to any user, whether or not the s1 byte is set, is then returned. If the dr byte is not set to '?', then the first match belonging to either user 0 or to the current user is returned.

If a match is made, then 00 hex is returned in register A, the dr byte of the FCB is set to the user number of the matched file, and the record on the disc containing the matched directory information is copied to the current DMA address. If there is no match made, then FF hex is returned in register A. In either case, the h2 byte of the FCB is cleared.

Function 18: Search For Next

Passed Values	Returned Values
None	Register A: Return Code

This function finds the next occurence of a file, following a previous call of a Search function; it may be used repetitively, but must have been preceeded by a usage of function 17 (Search For First), with no intervening file operations. The scan will continue from the last matched entry. The results are the same as with function 17.

Function 19: Delete File

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function removes all files whose FCB matches that addressed by registers DE. The match is made in the same way as in function 15 (Open File); the dr byte must not have the value '?'.

If a match was made and file(s) were deleted, then 00 hex is returned in register A. If no match was made, then FF hex is returned in register A.

Function 20: Read Sequential

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function reads the next record (in sequential ordering) from a file to the current DMA address. The FCB addressed by registers DE is used to refer to the file, which must have originally been Opened or Made. The cr byte is used to refer to the record being copied from the current extent. It is automatically incremented on each read; if it overflows, it is set to 00 hex and the next extent is entered.

If the read was successful, then 00 hex is returned in register A; if end of file occurs, then a non zero value is returned in register A.

Function 21: Write Sequential

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function writes the record at the current DMA address to a file. The FCB addressed by registers DE is used to refer to the file, which must have originally been Opened or Made. The cr byte is used to refer to the record of the current extent being written to; as in function 20 (Read Sequential), it is automatically incremented at each write. If it overflows, then the next extent is entered, and the cr field is reset to 00 hex (the first record of the new extent).

It should be noted that any records written to part of an already existent file will overwrite the old records.

If the write operation is successful, then 00 hex is returned in register A. If the operation is unsuccessful (e.g. a full disc) then a non zero value is returned in register A.

Function 22: Make File

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function creates a new file, and Opens it. The FCB addressed by DE is used to name the new file, and must therefore not already exist in the referenced disc directory. Use of function 19 (Delete File) before this function is adequate to ensure that this does not occur. TDOS will initialise the file directory and main memory value to indicate an empty file, create the file, and activate the FCB.

If the operation is successful, then 00 hex is returned in register A. If it is unsuccessful (e.g. no more directory space is available) then FF hex is returned in register A. In either case, the h2 byte of the FCB is cleared.

Function 23: Rename File

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return Code

This function renames all occurences of the file referred to by an FCB as the file named in the user field of the same FCB. As usual, the FCB is addressed by registers DE. The drive code for the user field is assumed to be 00 hex (default drive).

If the rename is successful, then 00 hex is returned in register A. If the FCB had no matches in the disc directory (and the rename was hence unsuccessful) then FF hex is returned in register A.

Function 24: Return Login Vector

Passed Values	Returned Values
None	Registers HL: Login vector

This function determines which drives are on-line by returning a login vector. The login vector value is a 16 bit value returned in registers HL, with the least significant bit of L referring to drive A, through to the most significant bit of H, which refers to drive P.

A zero indicates a drive that is off-line, whilst a one indicates that a drive is on-line, either as a result of being directly selected, or else as a result of a file operation specifying a non zero dr byte.

Function 25: Return Current Disc

Passed Values	Returned Values
None	Register A: Current Disc

This function returns the currently selected default disc in register A. The values range from 00 hex, corresponding to drive A, through to 0F hex, which corresponds to drive P.

Function 26: Set DMA Address

Passed Values	Returned Values
Registers DE: DMA address	None

This function is used to set the direct memory address (commonly known as the DMA), which is the address used to store records in, either after a read operation or before a write operation. The default DMA for CPN is 0080 hex, and on a cold or warm start, or a disc system reset, the DMA is set to this value. The DMA buffer should be one record, or 128 bytes, long.

Function 27: Get Address (Alloc)

Passed Values	Returned Values
None	Registers HL: Alloc address

This function returns the base address of the allocation vector for the currently selected disc drive. An allocation vector is stored in main memory for each on-line disc drive, and contains information useful for storage space calculations. Note that this information is likely to be inaccurate in the case of a read/only disc.

The allocation vector is a 32 byte long bit map, with each bit representing 16k bytes of store. There are a series of set bits, representing either space that is allocated, or space that is non-existent on the disc (i.e. the disc is smaller than 4M bytes). These are followed by a series of clear bits, representing the available space on the disc.

Function 28: Write Protect Disc

Passed Values	Returned Values
None	None

This function gives temporary write protect status for the currently selected disc, until the next cold or warm start. If an attempt is made to write to the disc, then the following message is output to the screen:

```
Disc d: Read Only!
```

Function 29: Get Read/Only Vector

Passed Values	Returned Values
None	Registers HL: R/O vector

This function returns a bit vector in registers HL, to indicate drives which have the temporary read/only bit set. As in function 24, the least significant bit of register L refers to drive A, and the most significant bit of register H refers to drive P. A one indicates read/only, a zero read/write.

Function 30: Set File Attributes

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return code

This function can be used to set the top bits of bytes f1-f8 and t1-t3 of an FCB; this is particularly useful for the read/only and system bits (t1 and t2 top bits respectively). A search is made for a match for the FCB addressed by DE (which should be unambiguous), ignoring the values of the top bits; the matched FCB is then changed to be that addressed by DE.

The top bits of bytes f1-f4 are available to the user; bytes f5-f8 and t3 are reserved for future system expansion.

If a successful match is made in the search, 00 hex is returned in register A; otherwise FF hex is returned in register A.

Function 31: Get Address (Disc Parms)

Passed Values	Returned Values
None	DPB address

This function returns the base address of the TCK resident disc parameter block in registers HL. It has two main uses: the values in the block are of use for display and space computation purposes; and when the disc environment changes, a transient program can dynamically change these values.

The Disc Parameter Block for the Torch contains the following values:

```
    00 01 07 7F 0F 00 FF 00 FF 80 00 00 00 00 00
```

It is provided for CP/M compatibility only.

Function 32: Set/Get User Code

Passed Values	Returned Values
Register E: FF hex (get)	Register A: Current user code
or User code (set)	or None

This function can be used either to find the current user code, or to change the user code. If register E has the value FF hex on entry, then the value of the current user number is returned in register A. Otherwise, the current user number is set to the value of register E (modulo 32).

Function 33: Read Random

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return code

This function is used to read a random record, selected by a 17 bit value held in the bytes r0-r2. Byte r0 is the least significant, whilst byte r2 contains the most significant (17th) bit. Normally, only bytes r0 and r1 are used, giving an index from 0 to 65535; byte r2 is only used to compute file size for function 35. Note that r2 must otherwise be zero, as a non zero value is used to indicate overflow beyond the end of the file.

To use a file for random access, it must first be Opened. The required record number is then entered into bytes r0 and r1, and TDOS is called to read the record into the buffer starting at the DMA. The record number, in contrast to sequential reading, is not updated with each operation; however, the ex and cr bytes are set with each read to the values for the record . It is therefore possible to read sequentially, commencing from a randomly accessed record; but note that on a change from random to sequential access, the same record will be read/written twice.

Upon successful completion of the operation, the value 00 hex is returned in register A. If the operation is not successful, the following values are returned: 01 hex indicates that an unwritten record has been accessed, and 06 hex indicates that byte r2 is non zero (i.e. an attempt has been made to read beyond the end of the disc).

Function 34: Write Random

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return code

This function writes a record of data from the DMA address to the disc. As in function 33 (Read Random), the bytes r0-r2 are not updated, but the ex and cr bytes are set. The returned values are the same as in function 33.

Function 35: Compute File Size

Passed Values	Returned Values
Registers DE: FCB address	Random record field of FCB set

This function returns the record address immediately after the end of a file, selected by the FCB addressed by registers DE. This is known as the virtual size of the file; if it has been written sequentially, it is the same as the physical size. If instead it has been written randomly, with some sequential disc space not allocated to the file, then the file may contain fewer records than indicated (e.g. if a file only contained record 65535 written in random mode, the size of the file would be given as 65536 records, although it only contained the one record).

The function searches for a match for the FCB addressed by registers DE. If byte r2 is set to Ø1 hex, then the file contained the maximum of 65536 records. Otherwise, bytes rØ and r1 contain the file size as a 16 bit value, with rØ as the least significant byte.

Function 36: Set Random Record

Passed Values	Returned Values
Registers DE: FCB address	Random record field of FCB set

This function returns the random record to which a file has been sequentially read/written. This has two main uses: either to produce a look-up table of the position of various keys in a sequentially read file, or to change between random and sequential reading. The file is identified by an FCB, addressed by registers DE. Bytes rØ and r1 of this FCB are set to the random record position last read/written.

Function 37: Reset Drive

Passed Values	Returned Values
Registers DE: Drive vector	Register A: Return code

This function resets specific disc drives, as indicated by a bit map in registers DE. The least significant bit indicates drive A, while the most significant bit corresponds to drive P. A 1 shows that a drive is to be reset.

A value of zero is returned in register A, unless a disc which was specified has open files on it that have been modified; in this case, a non zero result is returned.

Function 40: Write Random With Zero Fill

Passed Values	Returned Values
Registers DE: FCB address	Register A: Return code

This function is identical to function 34. It is provided for compatibility with systems which allocate records in groups, rather than singly.

THE BIOS VECTOR

High Memory in the Z8Ø is taken up by the BIOS vector, which is a series of jumps to routines useful to the programmer. These are:

FFCØ	JP USRIMM	Calls Torch Base User command (see section 4.4) given by byte following call.
FFC3	JP PUTIMM	Sends byte following call to 65Ø2 (see section 4.3)
FFC6	JP GETBYT	Return with Z8Ø register A containing byte received from 65Ø2.
FFC9	JP PUTBYT	Transmit the byte in Z8Ø register C to the 65Ø2.
BIOSV=		(Start of CP/M compatible BIOS vector)
FFCC	JP RESET	System reboot
FFCF	JP TWBOOT	System warmboot
FFD2	JP TGCONST	Get console status
FFD5	JP TGETKEY	Read in console character
FFD8	JP PCHARC	Write out console character
FFDB	JP TLSTCH	Print out character
FFDE	JP RETURN	(Reader output)
FFE1	JP RETURN	(Reader input)
FFE4	JP RETURN	(Seek track Ø)
FFE7	JP SELDSK	Select disc
FFEA	JP RETURN	(Set track)
FFED	JP RETURN	(Set sector)
FFFØ	JP SETDMA	Set DMA
FFF3	JP RETURN	(Read sector)

FFF6	JP RETURN	(Write sector)
FFF9	JP LISTST	Printer status
FFFC	JP RETURN	(Translate sector)
FFFF	DB VERSION	Version number

JP RETURN is used above for functions that are not supported by CPN. The equivalent CP/M functions are given in brackets afterwards.

TORCH PROGRAMMERS' GUIDE: SECTION 3

TORCH VDU CONTROL CODES

3.0 CONTENTS

3.1 INTRODUCTION

3.1.0 Facilities

This is a guide to Torch VDU controls. They are intended for calling from a program, and are used for changing modes of Input/Output. They should always be used in preference to any of the facilities outlined in Section 4 (Use of the Torch Base Processor), since the Torch VDU controls will continue to be supported on any future upgrades of the Torch Base processor.

There are several different streams of input/output available, allowing for a choice of configuration. For the Torch, the streams normally used will be Super VDU, graphics, and printer, for text output to screen, graphics output, and printed output respectively.

Facilities available include the following:

Super VDU Stream

1) Selection of different I/O streams (e.g. printer, vdu)
2) Positioning of the screen of output over a larger page of display in vdu memory.
3) Window selection on a page of memory.
4) Cursor movement and editing, both by line and column.
5) Changing screen colour, mode and enhancement.
6) Definition of new character sets.

Graphics Stream

1) Selection of different I/O streams.
2) Positioning of the screen of output over a larger page of display in vdu memory.
3) Cursor movement, line drawing and triangle filling.
4) Changing screen colour and mode.

Printer Stream

1) Selection of different I/O streams.

3.1.1 Accessing Torch VDU Functions

To use any of the facilities outlined in this section, the Torch VDU program must first be loaded into the Torch from the systems disc provided when you purchase your Torch. The file is called 'SUPERVDU.COM' and is loaded in the same way as any .COM file; i.e, you type:

```
SUPERVDU                     if the disc is in the top drive, or

B:SUPERVDU                   if the disc is in the bottom drive.
```

The program is loaded into the Torch's memory, so you may then remove the disc.

Torch VDU functions are invoked, once the program is loaded, by outputting the character <escape> from a program, followed by a character to indicate the function, and possibly a set of arguments in addition. For instance, to move the cursor left from a BCPL program, whilst in Super VDU stream:

```
        AND move.cursor.left () BE
        $( LET ascii.esc = #x1B
           wrch (ascii.esc)
           wrch ('W')
        $)

        move.cursor.left ()
```

or in Z80 assembler, under CPN:

```
        ESCAPE     EQU        1BH
        SCROUT     EQU        2

        CURLEFT:   LD         E, ESCAPE
                   RST        0030H
                   DB         SCROUT
                   LD         E, 'W'
                   RST        0030H
                   DB         SCROUT
```

Arguments passed to these functions are always numbers. In the case of every function save 'Relative Cursor Address in Window' (where the number is a signed sequence of ASCII numerals) the number is a string of unsigned ASCII numerals. Numerals are defined as the characters '0' to '9'.

Each argument after the first number must be preceeded by a separator, this being a string of one or more non numeric characters. Separators are swallowed. The last argument must be followed by a terminator, this being a single non numeric character. This character is swallowed; if it is a (carriage return) then the next character is swallowed if it is a (line feed); otherwise it is output.

For instance, to clear a page and select the mode from BCPL:

```
AND clear.page.and.select.mode (page, mode) BE
$( wrch (ascii.esc)
   writef ("& %n %n*n", page, mode)
$)
```

Invalid values of page and mode are ignored by the Torch VDU control program, and so there is no need for error handling in the user program.

Coordinates used for the Super VDU stream are measured in characters, and are hence dependent on the mode the Torch's display is set to. A screen can be 80, 40 or 20 units wide; and 32 or 25 units high. The origin of the page has coordinates (1,1).

3.2 TORCH VDU FUNCTIONS: SUPER VDU STREAM

3.2.0 Super VDU Stream: Overview

A number of pages (at least 1) of a size depending on the implementation (always at least one screenful) reside in memory. The screen may be positioned on this page wherever it is desired, and will always show all characters on the page in the area it outlines. Windows (a rectangular area of text) may be defined anywhere on the page that is desired, and may be of any size.

There is always a window selected as the current window, and the cursor may not be moved out of it using these functions. There is no need for the screen to display the current window or the cursor, and the screen's movement is independent of the cursor's movement. All editing is done inside the window; thus only the portion of a line inside the current window may be deleted, and so on. Whenever a new window is selected, the cursor is moved to the last position it was at in that window (or the top left hand corner of the window if it has not yet been used).

Windows may be designated as scrolling or non-scrolling. With both types, if an attempt is made to move the cursor off the top or bottom of a window, either to write or as a cursor movement, the cursor wraps round from top to bottom of the column it is in. If the cursor is moved off the end or beginning of a line, it is moved to the beginning of the next line down, or to the end of the previous line respectively. The two windows only differ in their handling of a cursor moving left off the top of a window, or right off the end of a window. If the window is a scrolling window, then all text is scrolled by one line down or up the window respectively. The line the cursor is on is made blank, and the cursor moved to the opposite end of the line. If the window is a non-scrolling window, then the two points wrap around to each other (for horizontal movement only).

3.2.1 Current Implementation

Currently, there is only one page of size one screenful. A maximum of 10 windows may be defined at any one time.

3.2.2 Super VDU Stream: Function List

Available Torch VDU functions for the Super VDU stream are given in a list below, and then described in detail in the next section.

<escape>
followed by:

General

<escape>	Select Stream
<space>	Initialise
'!'	"Page Mode" On/Off

Screen Selection

'$'	Select Page
'%'	Clear Page
'&'	Clear Page and Select Mode
"'"	Position Screen Origin
'('	Pan Screen Up
')'	Pan Screen Right
'*'	Pan Screen Down
'+'	Pan Screen Left

Use of Windows

'0'	Define Window in Page
'1'	Select Window
'2'	Clear Window
'3'	Clear to End of Window
'4'	Clear to Start of Window
'<'	Relative Cursor Address in Window
'='	Home Cursor in Window
'>'	Absolute Cursor Address in Window

Character Deletion and Insertion

'D'	Clear Line
'E'	Clear to End of Line
'F'	Clear to Start of Line
'H'	Clear Characters Right
'I'	Clear Characters Left
'L'	Delete Column
'M'	Insert Column
'N'	Delete Line
'O'	Insert Line
'P'	Delete Character
'Q'	Insert Character

Cursor Movement

'T'	Cursor Up
'U'	Cursor Right
'V'	Cursor Down
'W'	Cursor Left

Colour Selection

'\'	Select Foreground Colour
']'	Select Background Colour
'^'	Define Colour Relationship

Enhancement

'`'	Set Enhancement
'a'	Add Enhancement
'b'	Remove Enhancement

Character Definition

'd'	Define Character

3.2.3 Super VDU Stream: Function Specifications

Select Stream

<escape> <escape> and:

	Arguments	Comment
	'B', stream number n	Add stream n to output list
or	'C', stream number n	Remove stream n from output list
or	'A', stream number n	Clear output list; select stream n
or	'a', stream number n	Select input stream n

This function is used to control the I/O devices used by a program, as selected by the arguments. For input, only one device may be selected (to avoid input from more than one program, data file, etc.0. For output, several devices may be selected at once, by constructing a list of devices. An argument of 'A' clears the list, and then selects the specified stream, leaving it on the list.

Valid stream numbers are:

0	Sink/Null	
1	Dumb terminal/keyboard	|
2	Popular terminal/keyboard	| These are
3	Super terminal/keyboard	| mutually
10	Graphics terminal/keyboard	| exclusive
20	Printer	
100	Parallel port 0	
110	Serial port 0	

Initialise

Arguments	Comment
<escape> <space>	

This function initialises the vdu functions as follows:

1) I/O streams are selected for keyboard and screen only.
2) "Page mode" is set to off.
3) The screen origin is set to 1,1 on page 1.
4) The cursor is set to 1,1 (home).
5) The screen is set to mode 3, in black and white.
6) All pages and windows are cleared from memory.
7) There is no enhancement.
8) All defined characters are cleared.
9) Debug is set to off.

"Page Mode" On/Off

Arguments	Comment
<escape> '!' and	
'1'	On
or '0'	Off

This function is used to switch "page mode". If it is on, all output to the screen stops scrolling after every screenfull, until <shift> is pressed; if it is off, scrolling occurs continually.

Select Page

Arguments	Comment

(escape) '$' and:

page number n

This function selects the page from which the current screen of output is displayed. A page number either of 0, or greater than the available number of pages, causes the function to have no effect. Both the screen origin and the current cursor position are unchanged.

Clear Page

Arguments	Comment

(escape) '%' and:

page number n

This function clears the specified page of memory, and all windows for that page. A page number of 0 causes the function to clear the current page; a page number greater than the available number of pages causes the function to have no effect. The screen origin is set to 1,1; and the cursor is homed (to 1,1).

Clear Page and Select Mode

Arguments   Comment

(escape) ’&’ and:

page number n, mode m

This function clears the page as above; it also selects the mode of screen display. Available modes are:

Mode	Graphics	Text	Colours
0	640 x 256	80 x 32	2
1	320 x 256	40 x 32	4
2	160 x 256	20 x 32	16
3		80 x 25	2
4	320 x 256	40 x 32	2
5	160 x 256	20 x 32	4
6		40 x 25	2
7		40 x 25	Teletext

The new mode will have the default colours displayed (see ’Select Foreground Colour’); if it is required to change the colour mapping, ’Define Colour Relationship’ (see later) should be used.

Position Screen Origin

Arguments   Comment

(escape) "’" and:

x, y

This function positions the screen origin at the given x,y coordinates on the current page. The cursor remains in its current position on the page. Coordinates of 0,0, or ones that would position some or all of the screen off the current page, cause the function to have no effect.

Pan Screen Up

Arguments	Comment

(escape) '(' and:

displacement

This function pans the screen up the page (i.e. the y origin of the screen is decreased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Pan Screen Right

Arguments	Comment

(escape) ')' and:

displacement

This function pans the screen to the right of the page (i.e. the x origin of the screen is increased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Pan Screen Down

Arguments	Comment

<escape> '*' and:

displacement

This function pans the screen down the page (i.e. the y origin of the screen is increased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Pan Screen Left

Arguments	Comment

<escape> '+' and:

displacement

This function pans the screen to the left of the page (i.e. the x origin of the screen is decreased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Define Window in Page

Arguments	Comment
(escape) '0' and:	
window number,	Must be unique (see below)
page number, x, y, width, height, scroll type	

A window is created at origin x, y on the given page, and having the specified height and width. The window is assigned a number; if this number has been used previously, then the window is redefined to the new shape (so long as the arguments are legal; see below). A scroll type of one indicates that the window is a scrolling window, and one of zero that it is a non-scrolling window (See section 3.3.0, Accessing Torch VDU Commands). All other scroll types cause undefined actions.

Window 0 is used in the functions below to refer to all of the current page, and so may not be redefined. If a window number is selected that is greater than the maximum permissible one, then the function has no effect. A page number of zero indicates the current page; if the page number is greater than the number of pages, then the function has no effect. An x or y value of zero indicates that the current x or y position of the cursor should be used. A width of zero indicates that the window should extend to the right of the page; a height of zero indicates that the page should extend to the bottom of the page. Should any of the above arguments specify a window some or all of which is off the page, then the function has no effect.

Select Window

Arguments	Comment
(escape) '1' and:	
window number	

The specified window is selected as the current window. The position of the screen is not affected. The cursor is moved to the last position it had when the new window was last current; if it has never been selected previously, then the cursor is homed to the top left of the window.

Window Ø always corresponds to the whole of the current page. If a window is selected that has not been defined, then the function has no effect.

Clear Window

Arguments	Comment
(escape) '2' and:	
window number	

The specified window is cleared (i.e. it is made blank). The window boundaries remain, and the cursor is homed to the top left of the window.

If window Ø is cleared, this function has the same effect as 'Clear Page' acting on the current page. If a window that has not been defined is passed, then the function has no effect.

Clear to End of Window

Arguments	Comment
(escape) '3'	

The current window is cleared, starting with the current cursor position, and clearing to the end of the current line and all subsequent lines. The cursor is not moved.

Clear to Start of Window

Arguments	Comment
(escape) '4'	

The current window is cleared, starting with the character to the left of the cursor, and clearing to the start of the current line and all previous lines. The cursor is not moved.

Cursor Address Relative in Window

Arguments Comment

<escape> ')' and:

x, y

The cursor is moved by the specified displacements in the current window. Arguments of 0, 0 or ones that would remove the cursor from the window cause the function to have no effect.

Home Cursor in Window

Arguments Comment

<escape> '='

The cursor is homed in the current window (moved to the defined origin).

Cursor Address Absolute in Window

Arguments Comment

<escape> '<' and:

x, y

The cursor is moved to the absolute address given, relative to the origin of the current window. Coordinates that would remove the cursor from the current window, or ones of zero make the function have no effect. Coordinates of 1,1 represent the origin of the current window.

Clear Line

Arguments	Comment

<escape> 'D' and:

line number

The specified line of text in the current window is cleared (i.e. all characters are replaced by spaces). A line number of zero indicates that the line the cursor is currently on is to be cleared. If a line number is given that is not in the current window, the function has no effect.

Clear to End of Line

Arguments	Comment

<escape> 'E'

The current line of text is cleared (i.e. all characters are replaced by spaces), from the current cursor position to the right hand edge of the current window (inclusive).

Clear to Start of Line

Arguments	Comment

<escape> 'F'

The current line of text is cleared (i.e. all characters are replaced by spaces), from the left hand edge of the current window to the character on the left of the cursor (inclusive).

Clear Characters Right

Arguments	Comment

(escape) 'H' and:

number of characters

The specified number of characters are cleared, starting with the current cursor position, and moving to the right hand edge of the current window. If more characters are specified than are on the right of the window, then the function has no effect; if zero characters are specified, then the function has the same effect as 'Clear to End of Line'.

Clear Characters Left

Arguments	Comment

(escape) 'I' and:

number of characters

The specified number of characters are cleared, starting with the character to the left of the current cursor position, and moving to the left hand edge of the current window. If more characters are specified than are on the left of the window, then the function has no effect; if zero characters are specified, then the function has the same effect as 'Clear to Start of Line'.

Delete Column

TO BE IMPLEMENTED

Arguments	Comment
(escape) 'L' and:	
column number	

The specified column is deleted from the current window; as a result all columns to the right of this are moved one column to the left. The rightmost column of the window is filled with blanks. A column number of zero indicates that the line the cursor is currently on is to be deleted, and a column number that is not in the current window causes the function to have no effect.

Insert Column

TO BE IMPLEMENTED

Arguments	Comment
(escape) 'M' and:	
column number	

All columns in the current window to the right of (and including) the specified column are moved one column to the right, and a blank column is inserted in the specified column. The contents (if any) of the rightmost column of the window are lost. A column number of zero indicates that the column the cursor is currently on is the column where insertion takes place; if the column number is not in the current window, then the function has no effect.

Delete Line

Arguments	Comment

(escape) 'N' and:

line number

The given line is deleted, and all lines below it on the current window are scrolled up one line.The bottom line of the window is filled with blanks. A line number of zero indicates that the current line is to be deleted; if the line number is not in the current window, the function has no effect.

Insert Line

Arguments	Comment

(escape) 'O' and:

line number

All lines from the specified line downwards are scrolled down one line in the current window, and a blank line is inserted. The bottom line of the window is lost. A line number of zero shows that the current line is the site of insertion; if the line number is not in the current window, then the function has no effect.

Delete Character

TO BE IMPLEMENTED

Arguments	Comment

(escape) 'P'

The character at the current cursor position is deleted, and all characters to the right of this on the current line and in the current window are moved one space to the left. The rightmost character of the window is filled with a space.

Insert Character

TO BE IMPLEMENTED

Arguments	Comment

(escape) 'Q'

All characters on the current line of the current window, to the right of (and including) the current cursor position, are moved one place to the right, and the current cursor position is filled with a space. The rightmost character of the window is lost.

Cursor Up

Arguments	Comment
<escape> 'T'	

The cursor is moved up the current window by one line. The screen is not scrolled. The cursor wraps around from the top of the window to the same column on the bottom line of the window.

Cursor Right

Arguments	Comment
<escape> 'U'	

The cursor is moved across the current window one column to the right. The screen is not scrolled. The cursor wraps around from the right of the window to the left hand side of the next line down. The action on attempting to pass the end of the window depends on the scrolling type of the window. If the window is non scrolling then the cursor is moved to the top left hand corner (the origin) of the window. If the window is a scrolling window then each line of text in the window is scrolled up one line (and the top line is lost). The bottom line is made blank, and the cursor moved to the left of it.

Cursor Down

Arguments	Comment

(escape) 'V'

The cursor is moved down the current window one of line. The screen is not scrolled. The cursor wraps around from the bottom of the window to the same column on the top line of the window.

Cursor Left

Arguments	Comment

(escape) 'W'

The cursor is moved across the current window one column to the left. The screen is not scrolled. The cursor wraps around from the left of the window to the right hand side of the next line down. The action on attempting to pass the start of the window depends on the scrolling type of the window. If the window is non scrolling then the cursor is moved to the bottom right hand corner of the window. If the window is a scrolling window then each line of text in the window is scrolled down one line (and the bottom line is lost). The top line is made blank, and the cursor moved to the right of it.

Select Foreground Colour

Arguments   Comment

(escape) '\' and:

colour code

The selected colour is used for the foreground of the screen. Normal default codes are:

Two colour modes

0 = Black
1 = White

Four colour modes

0 = Black
1 = Red
2 = Yellow
3 = White

Sixteen colour modes

0 = Black
1 = Red
2 = Green
3 = Yellow
4 = Blue
5 = Magenta
6 = Cyan
7 = White
8 = Flashing Black/White
9 = Flashing Red/Cyan
10 = Flashing Green/Magenta
11 = Flashing Yellow/Blue
12 = Flashing Blue/Yellow
13 = Flashing Magenta/Green
14 = Flashing Cyan/Red
15 = Flashing White/Black

These default definitions may be changed by 'Define Colour Relationship' (see below).

Select Background Colour

Arguments   Comment

(escape) ']' and:

colour code

This function defines the background colour of the screen, in the same way as 'Select Foreground Colour' defines the foreground colour (see above).

Define Colour Relationship

Arguments Comment

<escape> '^' and:

logical colour code,
physical colour code

The given logical colour code, modulo the number of colours in the current mode, is redefined for the current mode of screen, according to the physical colour code given (which is the same as the default logical codes for sixteen colour modes; see 'Select Foreground Colour'). All colour relationships apply only to the current mode, and are cleared when the mode is changed.

It should be noted that redefining logical colour 7 will always change the foreground, and redefining logical colour 0 will select the background colour.

Set Enhancement

Arguments Comment

<escape> '`' and:

enhancement mode

The list of enhancement modes is cleared and set to the enhancement passed as an argument. This new list is used as the mode of enhancement used for all enhanced text until changed.

Currently available modes of enhancement are:

0	No enhancement
2	Inverse
5	Underlined

Mode 0 may not be used to add to a list of enhancements (and hence may not be usefully removed.)

Add Enhancement

Arguments	Comment

<escape> 'a' and:

enhancement mode

The specified enhancement mode is added to the list of enhancement modes, and the new list is used whenever enhanced text is written.

Remove Enhancement

Arguments	Comment

<escape> 'b' and:

enhancement mode

The enhancement mode passed to the function is removed from the list of enhancement modes, and the new list is used whenever enhanced text is written.

Character Definition

Arguments	Comment
<escape> 'd' and:	
character code,	Normally ASCII
character width,	| In
character height,	| pixels
row representation(s)	

The character code given is redefined to produce the character programmed by the user. The character code is given, followed by the width and height of the new character to be generated.

The representation of each row of pixels is generated as follows: each row, working from top to bottom, is written as a binary number, with each bit representing a pixel; a one indicates that the pixel is in the foreground colour, a zero that it is in the background colour. The binary number is then converted to an ASCII decimal number (unsigned), and passed to the function.

Debug On/Off

Arguments	Comment
<escape> <delete> and:	
1	On
or 0	Off

If debug mode is selected, by giving an argument of 1, then all output is passed directly, and VDU control codes are not invoked. If debug is switched off, then VDU codes are invoked.

3.3 TORCH VDU FUNCTIONS: GRAPHICS STREAM

3.3.0 Graphics Stream: Overview

A number of pages (at least one) of a size depending on the implementation (always at least one screenful) reside in memory. The screen may be positioned on this page anywhere that is desired. Unlike the Super VDU stream, there is no windowing.

Routines exist for moving the cursor around the screen, either leaving no trail, or drawing a line behind itself, or filling in triangles as it goes. There are comprehensive facilities for changing the colours used to plot.

It should be noted that all routines for moving the screen use the same system of coordinates as in the Super VDU stream; that is, the screen is 80, 40 or 20 units wide and 25 or 32 units high, depending on the mode; and the origin is the top left of the page (1,1). Movement of the cursor is done using graphics coordinates; the screen is 1280 units wide and 1024 units high, and the origin is at the bottom left of the page (0,0).

3.3.1 Current Implementation

There is currently one page of size one screenful.

3.3.2 Graphics Stream: Function List

Available Torch VDU functions for the Graphics Stream are given in a list below, and then outlined in detail in the next section.

<escape>
followed by:

General

<escape>	Select Stream
<space>	Initialise

Screen Selection

'$'	Select Page
'%'	Clear Page
'&'	Clear Page and Select Mode
"'"	Position Screen Origin
'('	Pan Screen Up
')'	Pan Screen Right
'*'	Pan Screen Down
'+'	Pan Screen Left

Graphics

'h'	Move Cursor Relative
'i'	Move Cursor Absolute
'j'	Draw Relative
'k'	Draw Absolute
'm'	Plot
'o'	Define Graphics Origin

Colour Selection

'\\'	Select Graphics Foreground Colour
']'	Select Graphics Background Colour
'^'	Define Graphics Colour Relationship

Debug

<delete>	Debug On/Off

3.3.3 Graphics Stream: Function Specifications

Select Stream

<escape> <escape> and:

	Arguments	Comment
	'B', stream number n	Add stream n to output list
or	'C', stream number n	Remove stream n from output list
or	'A', stream number n	Clear output list; select stream n
or	'a', stream number n	Select input stream n

This function is used to control the I/O devices used by a program, as selected by the arguments. For input, only one device may be selected (for obvious reasons). For output, several devices may be selected at once, by constructing a list of devices. An argument of 'A' clears the list, and then selects the specified stream, leaving it on the list.

Valid stream numbers are:

0	Sink/Null	
1	Dumb terminal/keyboard	|
2	Popular terminal/keyboard	| These are
3	Super terminal/keyboard	| mutually
10	Graphics terminal/keyboard	| exclusive
20	Printer	
100	Parallel port 0	
110	Serial port 0	

Initialise

Arguments	Comment

<escape> <space>

This function initialises the vdu functions as follows:

1) I/O streams are selected for screen and vdu only.
2) The screen origin is set to 1,1 on page 1.
3) The cursor is set to 0,0 (the bottom left of the page).
4) The screen is set to mode 0, in black and white.
5) All pages are cleared from memory.
6) The graphics origin is set to 0,0.
7) Debug is set to off.

Select Page

Arguments	Comment

<escape> '$' and:

page number n

This function selects the page from which the current screen of output is displayed. A page number either of 0, or greater than the available number of pages, causes the function to have no effect. Both the screen origin and the current cursor position are unchanged.

Clear Page

Arguments	Comment

<escape> '%' and:

page number n

This function clears the specified page of memory. A page number of Ø causes the function to clear the current page; a page number greater than the available number of pages causes the function to have no effect. The screen origin is set to 1,1; and the cursor is homed (to Ø,Ø).

Clear Page and Select Mode

Arguments	Comment

<escape> '&' and:

page number n, mode m

This function clears the page as above; it also selects the mode of screen display. Available modes are:

Mode	Graphics	Text	Colours
Ø	64Ø x 256	8Ø x 32	2
1	32Ø x 256	4Ø x 32	4
2	16Ø x 256	2Ø x 32	16
4	32Ø x 256	4Ø x 32	2
5	16Ø x 256	2Ø x 32	4

The new mode will have the default colours displayed (see 'Select Graphics Foreground Colour'); if it is required to change the colour mapping, 'Define Graphics Colour Relationship' (see later) should be used.

Position Screen Origin

Arguments   Comment

<escape> "'" and:

x, y

This function positions the screen origin at the given x,y coordinates on the current page. The cursor remains in its current position on the page. Coordinates of 0,0, or ones that would position some or all of the screen off the current page, cause the function to have no effect.

Pan Screen Up

Arguments   Comment

<escape> '(' and:

displacement

This function pans the screen up the page (i.e. the y origin of the screen is decreased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Pan Screen Right

Arguments **Comment**

(escape) ')' and:

displacement

This function pans the screen to the right of the page (i.e. the x origin of the screen is increased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Pan Screen Down

Arguments **Comment**

(escape) '*' and:

displacement

This function pans the screen down the page (i.e. the y origin of the screen is increased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Pan Screen Left

Arguments Comment

<escape> '+' and:

displacement

This function pans the screen to the left of the page (i.e. the x origin of the screen is decreased) over the specified displacement. If the specified displacement is zero, or if it would move some or all of the screen off the page, the function has no effect.

Move Cursor Relative

Arguments Comment

<escape> 'h' and:

x, y

The cursor is moved by the specified displacement from its current position. Coordinates of 0,0, or ones that would remove the cursor from the page, cause the function to have no effect.

The function is the same as Plot 0, x, y.

Move Cursor Absolute

Arguments	Comment

(escape) 'i' and:

x, y

The cursor is moved to the coordinates given, relative to the defined graphics origin. If the coordinates passed are off the page, then the function has no effect.

The function is the same as Plot 4, x, y.

Draw Relative

Arguments	Comment

(escape) 'j' and:

x, y

The cursor draws a straight line across the screen in the graphics foreground colour, moving by the specified displacement relative to its current position. If the cursor would be removed from the screen, then the function has no effect.

The function is the same as Plot 1, x, y.

Draw Absolute

Arguments — Comment

<escape> 'k' and:

x, y

The cursor draws a straight line across the screen in the graphics foreground colour, moving to the coordinates given (relative to the current graphics origin). If the cursor would be removed from the page, then the function has no effect.

The function is the same as Plot 5, x, y.

Plot

Arguments   Comment

<escape> 'm' and:

Plot code, x, y

Plot is used to draw points, lines and triangles to the screen, according to the plot code given. These are given in the list below:

0 Move relative to last point

1 Draw line relative to last point in graphics foreground colour

2 Draw line relative to last point in logical inverse colour

3 Draw line relative to last point in graphics background colour

4 Move to absolute position

5 Draw line from last point to absolute position in graphics foreground colour

6 Draw line from last point to absolute position in logical inverse colour

7 Draw line from last point to absolute position in graphics background colour.

8-15 As 0-7, but with the last pixel on the line not filled.

16-23 As 0-7, but with a dotted line instead of a solid one.

24-31 As 0-7, but with a dotted line, and with the last pixel on the line not filled.

64-71 As 0-7, but only the last pixel on any line is filled.

80-87 As 0-7, but plot and fill a triangle. The last two points visited are joined with the specified point to form a triangle, and it is filled.

All other values are reserved for future expansion.

'Relative to last point' means moving by the given x, y coordinates from the last point visited. An 'absolute position' is one given as coordinates relative to the defined graphics origin.

The logical inverse to a colour is (highest logical colour code for current mode) - (logical colour code); e.g. for a four colour mode:

logical	inverse
0	3
1	2
2	1
3	0

Define Graphics Origin

Arguments **Comment**

(escape) 'o' and:

x, y

The graphics origin is redefined to be at the given x, y coordinates, relative to the default origin of 0,0 at the bottom left hand corner of the page.

Select Graphics Foreground Colour

Arguments Comment

(escape) '\' and:

colour code

The selected colour is used for the foreground of the screen. Normal default codes are:

Two colour modes

0 = Black
1 = White

Four colour modes

0 = Black
1 = Red
2 = Yellow
3 = White

Sixteen colour modes

0 = Black
1 = Red
2 = Green
3 = Yellow
4 = Blue
5 = Magenta
6 = Cyan
7 = White
8 = Flashing Black/White
9 = Flashing Red/Cyan
10 = Flashing Green/Magenta
11 = Flashing Yellow/Blue
12 = Flashing Blue/Yellow
13 = Flashing Magenta/Green
14 = Flashing Cyan/Red
15 = Flashing White/Black

These default definitions may be changed by 'Define Graphics Colour Relationship' (see below).

Select Graphics Background Colour

Arguments Comment

(escape) ']' and:

colour code

This function defines the background colour of the screen, in the same way as 'Select Graphics Foreground Colour' defines the foreground colour (see above).

Define Graphics Colour Relationship

Arguments	Comment
<escape> '^' and:	
logical colour code, physical colour code	

The given logical colour code, modulo the number of colours in the current mode, is redefined for the current mode of screen, according to the physical colour code given (which is the same as the default logical codes for sixteen colour modes; see 'Select Graphics Foreground Colour'). All colour relationships apply only to the current mode, and are cleared when the mode is changed.

It should be noted that redefining logical colour 7 will always change the foreground, and redefining logical colour 0 will select the background colour.

Debug On/Off

Arguments	Comment
<escape> <delete> and:	
1	On
or 0	Off

If debug mode is selected, by giving an argument of 1, then all output is passed directly, and VDU control codes are not invoked. If debug is switched off, then VDU codes are invoked.

TORCH PROGRAMMERS' GUIDE: SECTION 4
=====================================

USE OF THE TORCH BASE PROCESSOR
================================

4.0 CONTENTS

4.1 INTRODUCTION

4.1.0 The Torch Base Processor

The Torch Base processor is a peripheral processor, constructed using the main p.c.b. from an Acorn B.B.C. Microcomputer. A 6502 is used as the c.p.u. A large number of functions are available to the programmer using this board, or using this board in conjunction with the main Z80 c.p.u.

These features include a large number of functions providing the same functions as the CPN operating system, and facilities normally available on the B.B.C. microcomputer.

4.1.1 Future Upgrades

All the features given in this section are dependent on the Torch Base periheral processor. Most of them are available either from CPN, or from Torch VDU control codes, or by other methods.

Torch Computers Ltd. reserves the right to change the Torch Base peripheral processor in future upgrades to the Torch Computer. Although these features are documented for the convenience of programmers, no undertaking is made to continue to support them for future versions of the Torch. Wherever possible (which will be the majority of cases) the other features outlined in this guide should be used to achieve the end effect required.

4.2 TECHNICAL TERMS

Cutdown FCB: This is 12 bytes long. The first byte is the drive code, with Ø representing drive A, 1 representing drive B, and so on. The next 11 bytes are the ASCII representation of the file name and type. Note that this is NOT the same as taking the first 12 bytes of a CPN FCB (see section 2.4, File Control Blocks), since the drive code for the Torch Base is one less than that used in CPN.

File Handle: Each disc may have up to 256 directory entries. A file handle identifies a particular file by giving its position in the directory (from Ø to 255).

Cold Boot: The 6502 is reset by pressing the master reset button. Any action it is taking is halted. Note that this can be dangerous if, for instance, the 6502 is in the middle of writing a disc track. All Z8Ø memory will be uninitialised, save for CPN and CCCP, which are reloaded from ROM.

Warm Boot: CPN and CCCP are reloaded into Z8Ø memory. All remaining Z8Ø memory is uninitialised. The 6502 is not affected.

Hobnailed Boot: The message:

'User Program Error nn'

is displayed, where nn is a number having one of the following meanings:

01 The Z80 has issued an invalid command.

02 An attempt was made to read a record from a file that was not open.

03 An attempt was made to write a record to a file that was not open.

06 or A8 Invalid user function attempted.

55 Invalid file handle specified (usually due to using an FCB for transput without opening the corresponding file.

66 The 6502 failed to find enough store to load a file structure block.

77 A disc transfer was incomplete (disc controller timeout).

Results 66 and 77 are unlikely to occur unless disc controller parameters ar set up using peek and poke (see Systems Manual).

Following the message, a warm boot is performed.

Soft Boot: CCCP is reloaded from ROM into Z80 RAM. All other contents of the Z80 RAM are preserved. The 6502 is unaffected.

4.3 TORCH BASE COMMAND INTERFACE

4.3.0 Accessing Torch Base Commands

The Torch Base commands are a low level (below CPN and Torch VDU) interface with the Torch Base processor. A typical command will consist of the Z80 sending the number of the desired function to the 6502; there will then usually be an exchange of information between the Z80 and the 6502, occurring in the order given in each function specification. Note that in the first section of each specification, in which the values to be exchanged are given, there is no mention of the order in which they are to be passed; this is given in each description below. Note also that in some cases, not all the values given are passed.

For instance, if it was wanted to Peek Into RAM (the 6502's) at location 8000 hex, then the Z80 would send:

```
    0D  hex          (command  number)
    00  hex
    80  hex          (the  address  as  a low/high byte pair)
```

The 6502 would reply by sending the contents of address 8000 hex to the Z80.

The bytes that are passed are either sent by the Z80, or received by it. There is no user control over when the 6502 sends a byte (being a parallel processor, it does so as soon as it has completed a task), merely over when the Z80 receives that byte.

There are three routines available to send and receive bytes; these are located in the Torch BIOS vector, at fixed positions in memory. They are accessed as follows:

CALL 0FFC3H   Sends the byte following the call to the 6502.
               All Z80 registers apart from AF are preserved.

CALL 0FFC6H   Returns with a byte in Z80 register A received
               from the 6502.

CALL 0FFC9H   Sends the byte in Z80 register C to the 6502.
               Z80 registers AF are corrupted.

Throughout this manual the routines rx and tx respectively are used to represent the last two routines above.

4.3.1 Command List

0	Warm Boot
1.	Print Byte
2.	Open File
3.	Close File
4.	Search For First
5.	Search For Next
6.	Delete File
7.	Read Random
8.	Write Random
9.	Make File
A.	Rename File
B.	Set File Attributes
C.	Compute File Size
D.	Peek Into RAM
E.	Poke Into RAM
F.	Call User Command
10H (G)	Set/Get User Code
11H (H)	Get Keyboard Status/Input
12H (I)	Select Input
13H (J)	Select Output
14H (K)	Control Communications
15H (L)	Console Output
16H (M)	Get Input Status
17H (N)	Compute Extent Size

4.3.2 Command Specifications

Command 0: Warm Boot

Passed Values	Returned Values
None	None

On receiving this command, the 6502 issues a reset pulse to the Z80, initialises all internal tables for CPN, puts a startup message on the screen, initialises the wire interface, and waits for a "ready" byte from the Z80.

The 6502 then waits for further commands.

See description of Warm Boot for more information.

Command 1: Print Byte

Passed Values	Returned Values
ASCII character to be printed	None

The 6502 waits for an ASCII character from the Z80, which is added to the queue for printer output on the currently selected printer stream.

Command 2: Open File

Passed Values	Returned Values
Cutdown FCB	Return code, file handle, file opened.

The 6502 receives a cutdown FCB (as described earlier). Depending on the disc qualifier, the appropriate disc directory is searched for the first file with a name matching the FCB.

If a file is not found, a return code of FF hex is sent back; otherwise a return code of 00 hex is sent back. The information in the directory is then moved into a free slot in the file handle table, and a byte giving the position in the table is sent to the Z80, followed by the file name of the file just opened.

Should the file handle table be full, the effect is the same as if the file is not found.

If a file has not been closed since the last open command, the effect is to return the same handle as the last open command.

Command 3: Close File

Passed Values	Returned Values
Cutdown FCB	Return code

A cutdown FCB is accepted from the Z80, and the first file matching the FCB is located. The file handle table is then scanned to find the entry corresponding to the matched file. On finding this entry, the up-to-date information in the handle is copied into the directory entry on disc, and the handle entry is marked as free.

A return code of 00 hex indicates the file was successfully closed, or had not been opened; FF hex indicates failure to close the file.

Command 4: Search For First

Passed Values	Returned Values
Cutdown FCB	Return code, file name, user number.

The directory of the specified disc is scanned sequentially until a match is found for the cutdown FCB accepted from the Z80.

On finding a match, a return code of 00 hex is issued, and the name of the matching file is sentback to the Z80, followed by the user number of the file matched.

If a file is not found matching the cutdown FCB, a return code of FF hex is returned.

Command 5: Search For Next

Passed Values	Returned Values
None	Return code, file name, user number.

The directory of one or both discs is scanned sequentially, starting from immediately after the last matched file, until a match is found for the last cutdown FCB accepted from the Z80.

The codes returned and actions taken are the same as for command 4 (Search For First Match).

Command 6: Delete File

Passed Values	Returned Values
Cutdown FCB	Return code

This command accepts a cutdown FCB from the Z80, finds all matching files and removes them and all associated records and file structure from the disc.

If one or more files were deleted, a return code of 00 hex is issued by the 6502; otherwise a return code of FF hex is issued.

Command 7: Read Random

Passed Values	Returned Values
File handle, record number	Return code, record

The Z80 sends the 6502 a file handle; the first open file found that matches this is read from. If the handle does not refer to an open file, the 6502 issues a hobnailed boot. Otherwise, the 6502 reads a word (2 bytes pair) from the Z80, and attempts to read the record at that position in the file.

If the record exists, the 6502 sends back a return code of 00 hex, and then the 128 bytes of the record. If the record has not yet been written, the return code 01 hex is issued, and no record is passed.

Command 8: Write Random

Passed Values	Returned Values
File handle, record number, record.	Return code

The Z80 sends the 6502 a byte, indicating which file handle to use. This is followed by a word (2 bytes) indicating a record number. The 6502 responds either with 00 hex, indicating success, in which case the Z80 sends 128 bytes of data to make up the record; or with FF hex, indicating failure to write the record.

The writing of a record may cause several disc accesses, because the file may have to be extended, and new space allocated. The last disc access is overlapped with Z80 processing.

If the file handle supplied by the Z80 does not refer to an open file, a hobnailed boot is issued.

Command 9: Make File

Passed Values	Returned Values
Cutdown FCB	Return code, file handle.

The Z80 sends a cutdown FCB to the 6502, specifying the name of a file which it wishes to create. If a file with a matching name already exists on the disc, or if the directory is full (it holds up to 256 names), or there is insufficient room on the disc for the file structure, a return code of FF hex is issued. Otherwise the name of the file is entered in the directory of the appropriate disc, and an empty file connected to this directory entry.

When the file has been created, it is opened, and return codes as for command 2 (Open File) are issued, along with (if appropriate) a file handle.

Note that the name of the opened file is not returned by the Make File command as it is in the Open File command.

Command 10: Rename File

Passed Values	Returned Values
Current cutdown FCB, new cutdown FCB.	Return code

The 6502 accepts a cutdown FCB from the Z80, and then a second cutdown FCB. A scan is made of the directories for the current user and for user 0 to find files matching the first FCB. Each of these files has its name changed to the second FCB, but the attribute bits are not altered.

If one or more files can be renamed, a return code of 00 hex is issued. If no matching files are found, a return code of FF hex is issued.

Command 11 (B): Set File Attributes

Passed Values	Returned Values
New cutdown FCB	Return code

The 6502 accepts a cutdown FCB from the Z80. A scan is made of the directories for the current user and for user 0 for files matching the FCB, ignoring the values of the top bits. Each matched file then has its directory name changed to that given by the FCB. The attribute bits may be altered.

If one or more file names are matched, then a return code of 00 hex is given; if no matching files could be found then a return code of FF hex is issued.

Command 12 (C): Compute File Size

Passed Values	Returned Values
Cutdown FCB	Return code, size (2 bytes)

The 6502 accepts a cutdown FCB from the Z80, and finds the first matching filename in the directory. A return code of FF hex is issued if no match was made. One of 01 hex is issued if the file is full (i.e. it has length 64k records). Otherwise, a return code of FF hex is issued, followed by a low-high pair of bytes giving the virtual length of the file.

The virtual length of a file is defined as the record immediately after the last record to be written; this will be the same as the physical file size if the file has been sequentially written. If, however, the file has been randomly written, with some sequential disc space not allocated to the file, then the file may contain fewer records than indicated. (If, for instance, only the 4000th record has been written to, the file length would be given as 4001, although only one record has been written.)

The virtual length of an empty file is zero.

Command 13 (D): Peek Into RAM

Passed Values	Returned Values
Address	(Address)

The Z80 sends a 2 byte address as a low/high pair of bytes. The 6502 returns the contents of that location as a single byte.

Command 14 (E): Poke Into RAM

Passed Values	Returned Values
Address, data	None

The Z80 sends a 2 byte address, as low then high bytes, followed by a single byte, which the 6502 pokes into that location.

Command 15 (F): Call User Command

Passed Values	Returned Values
User command number	

User commands are special commands which are provided for Torch systems work. Each command is called by the Z80 issuing the appropriate user command number.

The available user functions and their numbers are listed below, in section 4.4 (Torch Base User Command Interface), which also contains details of arguments and gives the specifications of these functions.

Command 16 (G): Set/Get User Code

Passed Values	Returned Values
FF hex (get)	Current user code
or User code (set)	None

The Z8Ø sends a single byte to the 65Ø2. If the byte is FF hex, then the 65Ø2 returns a byte giving the currently set user number. Otherwise it sets the user number to the byte passed by the Z8Ø.

The initial user number after warm boot is zero.

Command 17 (H): Get Keyboard Status/Input

Passed Values	Returned Values
None	Return code, ASCII character

The 65Ø2 returns either ØØ hex, indicating no console input is pending; or FF hex, followed by the ASCII code of the console input pending.

Codes other than ØØ hex or FF hex for the first byte are reserved for future use.

Command 18 (I): Select Input

Passed Values	Returned Values
Console input code	None

The byte sent by the Z80 following this command indicates which device should be treated as console input. The following selections are available, all others causing undefined actions:

0: Read keyboard, and lose RS423 input. (The RS 423 buffer is unaffected)

1: Lose keyboard input; if RS 423 buffer is empty, read from RS 423 input; otherwise read from buffer, and buffer RS 423 input.

2: Read keyboard; add RS 423 input to buffer.

Command 19 (J): Select Output

Passed Values	Returned Values
Console output code	None

The byte sent by the Z80 following this command indicates which device should be treated as console output. The following selections are available, all others causing undefined actions:

Screen output	0
Printer output	Not yet defined
RS423 output	3
Special output	Not yet defined

Command 2Ø (K): Control Communications

Passed Values	Returned Values
Communication Command Code	

This command is used to interface to the asynchronus communications channel on the Torch Base peripheral processor. A byte is sent following this command indicating what action should occur.

The following communications commands are defined so far:

ØØ - Poll communications interface
Ø1 - Perform interrupt action

After every normal CP/N command, a communications poll may be made. This is controlled by a byte at some fixed offset from the peripheral variable area.

For more information see TSM03 - Torch Communications Technical Manual.

Command 21 (1): Console Output

Passed Values	Returned Values
ASCII character	None

The Z8Ø sends a byte to the 65Ø2, which is sent down the console output channel.

Command 22 (M): Get Input Status

Passed Values	Returned Values
None	Return Code

The 6502 returns a byte indicating the console status. If no console input is pending, then 00 hex is issued; if input is pending, then FF hex is issued.

Note that the console input may be redirected.

Command 23 (N): Compute Extent Size

Passed Values	Returned Values
File handle, extent number	Number of records

The Z80 sends the 6502 a file handle to indicate the file to be used in the calculation. If the file handle does not refer to an open file then a hobnailed boot is issued. Otherwise, the Z80 sends the 6502 an extent number, and the 6502 returns the number of records in that extent (from 0 to 128).

4.4 TORCH BASE USER COMMANDS

4.4.0 Accessing User Commands

All user commands are accessed by calling Torch Base command 15 (Call User Command) and passing the 6502 the number of the user function desired. Thus

```
     tx 0FH, tx 00H
```

would call user command 0 (Read Block).

User commands may also be accessed through the BIOS vector, by calling location FFC0, with the byte following the call containing the call number, e.g:

```
     CALL      0FFC0H
     DB        00H
```

would also call User Command 0.

4.4.1 User Command List

0		Read Block
1		Write Block
2		Select Disc
3		Format Track
4		
5		Get Escape Status
6		Move 40 Bytes
7		Return Version Number
8		Initialise Disc
9		Unslave Disc Caches
10	(0A)	Select Debug State
11	(0B)	Get Character Hardware Definition
12	(0C)	Call Osword
13	(0D)	Read Scratchpad Byte
14	(0E)	Write Scratchpad Byte
15	(0F)	Call Osbyte
16	(10)	Toggle Printer
17	(11)	Get Output Status
18	(12)	Get Disc Error Count
19	(13)	Reset Drive
20	(14)	Get Disc Allocation Vector

4.4.2 User Command Specifications

User Command 0: Read Block

Passed Values	Returned Values
Block number	(Block)

The Z80 sends the 6502 a block number, and the 6502 returns the contents of that block from the current disc, as a sequence of 256 bytes. If the specified block number is invalid (i.e. does not exist on the disc) then a hobnailed boot is issued instead.

For more information, see the Systems Manual.

User Command 1: Write Block

Passed Values	Returned Values
Block number, block	None

The Z80 sends the 6502 a block number to be written to. if this number is invalid, then a hobnailed boot is issued. If it is valid, then the next 256 bytes sent by the Z80 are written to the specified block.

For more information, see the Systems Manual.

User Command 2: Select Disc

Passed Values	Returned Values
Drive code	None

The Z80 sends the 6502 a byte to indicate the drive to be selected. 00 hex represents drive A, 01 drive B, and so on to 0F hex which represents drive P on a 16 drive system.

User Command 3: Format Track

Passed Values	Returned Values
Track/side number	Return code

The Z80 sends the 6502 a byte, the least significant bit of which contains the side number, and the other seven bits of which contain the track number. If this value is illegal, a hobnailed boot is issued. Otherwise, the 6502 attempts to format the specified track; a return code of 00 hex indicates success. Other return codes (modulo 32) are:

08 - 0E	System error; recovery possible.
10 - 16	Operator error.
18 - 1E	Program/hardware error; generally fatal.

If a return code is over 32, then deleted data was found on the disc.

For more information, see the Systems Manual.

User Command 4

Passed Values	Returned Values
None	None

Reserved for future 6502 software development. See the Systems Manual.

User Command 6: Move 40 Bytes

Passed Values	Returned Values
Address	None

The Z80 sends the 6502 a low/high pair of bytes giving an address in 6502 memory. The next 40 bytes of data, starting at the given address, are moved 40 bytes towards high memory. If the address value would cause any of the bytes to pass off the top of memory, they are wrapped around to low memory.

This command is largely useful for the management of Prestel double height lines.

User Command 7: Return Version Number

Passed Values	Returned Values
None	Version number

The 6502 returns a byte representing the version of the 6502 Torch Base ROM currently installed. Note that this is not necessarily the same as the current version of CPN.

The byte 23 hex would be returned for version 2.3, 5A hex for version 5.10, and so on.

User Command 8: Initialise Disc

Passed Values	Returned Values
None	Return code

All tracks on the current disc are formatted, and an empty directory is written to track zero. A return code of 00 hex indicates success. Other return codes (modulo 32) are:

08 - 0E	System error; recovery possible.
10 - 16	Operator error.
18 - 1E	Program/hardware error; generally fatal.

If a return code is over 32, then deleted data was found on the disc.

For more information, see the Systems Manual.

User Command 9: Unslave Disc Caches

Passed Values	Returned Values
None	None

All disc information in the memory of the Torch Base processor that is more recent than that on disc is written to the disc. All disc information is then erased from Torch Base memory.

User Command 10: Select Debug State

Passed Values	Returned Values
Debug vector	None

The Z80 sends a single byte debug vector to the 6502, which controls both the debug mode in use, and the usage of the numeric keypad. If bit 7 is the most significant bit, then each bit has the following effect:

Bit	Effect
0	The top bit of the numeric key pad is set
1	The keys 1 - 9 move the cursor about the screen, horizontally, vertically or diagonally. '0' is defined as 'copy'.
2	Reserved for future expansion
3	Reserved for future expansion
4	Reserved for future expansion
5	When a disc is being used in a read or write operation, the block number being used is printed to the screen, with an 'r'for read or a 'w' for write.
6	On a disc error, there is no hobnailed boot. (This is reserved for test use only).
7	All bytes passed between the 6502 and the Z80 are traced on the screen.

A new debug vector cancels the effect of all previous debug vectors.

User Command 11: Get Character Hardware Definition

Passed Values	Returned Values
ASCII character code	Character definition

The Z80 sends a byte to the 6502, giving the ASCII character code whose definition is wanted. The 6502 returns a series of eight bytes, which defines the pixels in each row of the character from top to bottom. A bit that is set indicates that the pixel is in the foreground colour; one that is cleared indicates a pixel in the background colour.

Note that this call always returns the same results (i.e. the default character definitions that have been preprogrammed). To read any user defined characters, Osword Call 10 (Get Character Software Definition) should be used.

User Command 12: Call Osword

Passed Values	Returned Values
Osword call number	

This interfaces to the Acorn Machine Operating System Osword calls, the call being determined by the byte passed by the Z80. For more information of Osword calls, and the ways of using them, see section 4.5 (Osword Call Interface).

User Command 13: Read Scratchpad Byte

Passed Values	Returned Values
Byte number	(Byte)

The Z80 passes a byte number to the 6502, and this is used as a displacement (from 0 to 3F hex) from the start of the scratchpad (see section 4.5.1: The Torch Base Scratchpad). The byte in this address is returned to the Z80 by the 6502.

User Command 14: Write Scratchpad Byte

Passed Values	Returned Values
Byte number, byte	None

A byte number is passed to the 6502, and this is used as a displacement (from Ø to 3F hex) from the start of the scratchpad (see section 4.5.1: The Torch Base Scratchpad). The 6502 then accepts a byte, and stores it in the specified location in the scratchpad.

User Command 15: Call Osbyte

Passed Values	Returned Values
Osbyte call number, x, y.	x, y.

The Z8Ø passes the 65Ø2 an Osbyte call number, and then two parameters for that call. The 65Ø2 performs the call (see section 4.6: Osbyte Call Interface for a list of Osbyte calls and their specifications) and then returns a two byte result. The result may or may not contain useful information, depending on the call.

User Command 16: Toggle Printer

Passed Values	Returned Values
None	None

If the printing device was being used for output, it is no longer; if it was not being used for console output, then it is put in stream.

User Command 17: Get Output Status

Passed Values	Returned Values
None	Return Code

The 6502 issues a return code to the Z80; 00 hex indicates that the printer queue or the RS423 buffer is full, and FF hex indicates that they are both empty.

User Command 18: Get Disc Error Count

Passed Values	Returned Values
None	Disc Error Count

The 6502 sends the Z80 a count of the number of unsuccessful attempts to access a disc that have been made. The count is initialised on power up to an undefined value, and then incremented for every unsuccessful access. The value is passed as a two byte value, with the least significant byte first. Note that if an error code of 01 hex is issued by the 6502 during a disc access, the system will try to recover, and up to 10 attempts may be made to access the disc before the system stops.

User Command 19: Reset Drive

Passed Values	Returned Values
Disc vector	Return code

The Z80 passes the 6502 a bit map of discs to be reset. The first byte received is used for drives A - H (low - high bits respectively) and the second byte for drives I - P (low - high again). A bit that is set indicates that the drive is to be reset; one that is cleared shows the drive is not to be reset.

A return code is issued by the 6502: a value of 00 hex shows that the reset(s) were successful, whilst one of FF hex shows they were unsuccessful.

User Command 20: Get Disc Allocation Vector

Passed Values	Returned Values
Disc number	Allocation vector

The Z80 sends the 6502 a disc number, 00 hex corresponding to disc A, through to 0F hex for drive P on a 16 drive system. The 6502 returns a 32 byte value, being a representation of the amount of space used on the disc. One bit is used to represent 16k bytes on the disc; the value returned will be a stream of ones, indicating either that space is used, or was non-existent on the disc (it being smaller than 4M bytes) and then a stream of zeroes, indicating unused space.

Note that no information about disc layout is given or implied.

4.5 OSWORD CALL INTERFACE

4.5.0 Accessing Osword Calls

Osword calls are accessed by User Command 12, which is itself accessed by the Torch Base Command 15. The call number is passed by the Z80 to the 6502, after calling the above commands. Thus to call Osword call nn, the following calls are made:

```
     tx 0FH, tx 0CH, tx nnH.
```

Parameters for the Osword calls are passed in the scratchpad (for more information, see the next section, The Torch Base Scratchpad), and results (if any) are also returned in it. More detailed information is given in the specification of each call. When the call is made, the system sets the XY register pair of the 6502 to point at the base of the scratchpad, and the Osword call number is placed in register A.

4.5.1 The Torch Base Scratchpad

The 6502 has a 40H area of memory, known as the scratchpad, which is used both for passing parameters to and from Osword calls, and for use by programs running on the parasite processor. Bytes are read from the scratchpad using user call 13, and are written to the scratchpad using user call 14. Thus to read the value mmH of the nnth byte of the scratchpad, the following calls are made:

```
tx 0FH, tx 0DH, tx nnH,       rx mmH.
```

And to write the value mmH to the nnth byte of the scratchpad, the following calls are made:

```
tx 0FH, tx 0EH, tx nnH, tx mmH.
```

The first 16 bytes of the scratchpad are used for passing Osword call parameters, and may be overwritten during any Osword call. The allocation map of the scratchpad is as follows:

00 - 0F hex	Osword parameter block
10 - 3E hex	Available for general use
3F hex	Reserved for use in help status

4.5.2 Osword Call List

00	Pass Scratchpad to CLI
01	Read Absolute Time
02	Write Absolute Time
03	Read Interval Time
04	Write Interval Time
05	
06	
07	Make Sound
08	Define Envelope
09	Read Pixel
0A	Get Character Software Definition
0B	Read Colour Relationship
0C	
0D	Read Graphics Cursor Position

4.5.3 Osword Call Specifications

Osword Call 0: Pass Scratchpad to CLI

Passed Values	Returned Values
Line to be interpreted	

The contents of the scratchpad are passed to the Command Line Interpreter (CLI). If there is a valid '* Command' in the scratchpad, then the CLI will execute that command. Useful commands are:

*BASIC	Enters B.B.C. Basic
*KEY	Redefines soft keys
(*FX	Calls Osbyte; better done direct. See next section.)

For more details of these commands, see section 4.9 ('*' Commands).

Osword Call 1: Read Absolute Time

Passed Values	Returned Values
None	Absolute Time

This call reads the value of the absolute timer into bytes 00 - 04 of the scratchpad, with the least significant byte in byte 00. The absolute timer is an internal clock on the 6502, which takes the form of a counter, and counts upwards at a rate of one digit per 10 ms. It is zeroed whenever the system has a hard boot issued to it.

Osword Call 2: Write Absolute Time

Passed Values	Returned Values
Absolute Time	None

This call writes the value of bytes 00 - 04 of the scratchpad to the absolute timer, byte 00 being the least significant one. The absolute timer is an internal clock on the 6502, which takes the form of a counter, and counts upwards at a rate of one digit per 10 ms. It is zeroed whenever the system has a hard boot issued to it.

Osword Call 3: Read Interval Time

Passed Values	Returned Values
None	Interval Time

This call reads the value of the interval timer into bytes 00 - 04 of the scratchpad, with the least significant byte in byte 00. The interval timer is an internal clock on the 6502, which takes the form of a counter, and counts downwards at a rate of one digit per 10 ms. It is zeroed whenever the system is reset.

Osword Call 4: Write Interval Time

Passed Values	Returned Values
Absolute Time	None

This call writes the value of bytes 00 - 04 of the scratchpad to the interval timer, byte 00 being the least significant one. The interval timer is an internal clock on the 6502, which takes the form of a counter, and counts downwards at a rate of one digit per 10 ms. It is zeroed whenever the system is reset.

Osword Call 5

This call is reserved for future expansion.

Osword Call 6

This call is reserved for future expansion.

Osword Call 7: Make Sound

Passed Values	Returned Values
Sound Parameters	None

8 bytes are read from the scratchpad, and used to define a sound. They come as 4 low/high byte pairs, (known hereafter as words) and are used respectively (from byte 00 of the scratchpad) to define channel, amplitude/envelope, pitch and duration.

The sound parameters define the sound for one of three sound generating channels (numbered 1 -3) or a noise/pulse wave channel (numbered 0). Which channel is defined is indicated by the least significant hex digit (4 bits) of the channel word, which should be in the range 0 -3. Up to four definitions of sound may be queued in the sound buffer for each channel, in addition to the sound being played. The next least significant digit (byte 0, top bits) determines whether the sound is queued; if it is a one, then the sound buffer is flushed and the sound immediately output; if it is a zero, then the definition is queued. The second most significant digit (byte 1, low bits) is used for chord synchronisation, and indicates the number of other sound channels having the same value of this digit which must be ready before the chord was played. It should have a value between 0 (the normal default) and 3. The most significant digit (byte 1, top bits) is used to send a dummy sound to the channel, so that the previous sound will continue; this is only needed for some software applications. A one indicates that the note is a dummy, a zero that it is to be played.

Normally, only the channel is indicated; the other commonly used digit is that for chord synchronisation.

The amplitude word is taken as a five bit 2's complement number, the value being in the least significant bits (byte 2, low bits). A number from -16 to -1 is used for a note of constant pitch and amplitude to indicate the amplitude. A number from 0 to 15 is used to identify an envelope (See Osword call 8; Define Envelope) for use with sounds of varying pitch and amplitude.

The pitch word is a single byte value between 0 and 255, contained in the least significant byte (byte 4). A pitch of 0 represents the B 13 semitones below middle C. The pitch increases at a rate of one quarter semitone per digit. For the noise channel (channel 0), if bit 2 is set then it will produce grey noise; otherwise it produces a pulse wave. Bits 0 and 1 control the frequency of the wave; if they are both set the frequency will be linked to that of channel 1.

Finally, the duration is a single byte value between 0 and 255, also contained in the least significant byte. A duration of 255 will give a note without end; otherwise, the value is in units of 50ms.

Osword Call 8: Define Envelope

Passed Values	Returned Values
Envelope Parameters	None

The 6502 is passed 14 bytes in the scratchpad, and defines both a frequency envelope and an amplitude envelope from them.

Byte 0 of the scratchpad is used to define the envelope number (from 0 to 15), as used in the Make Sound call (see above). Byte 1 is used for the time interval (from 1 to 127) at which the envelopes are updated, in units of 10ms. If the top bit of this value is set, then the frequency envelope is retriggered after each cycle until the amplitude envelope reaches zero, or the duration of the note expires; if it is cleared, then the frequency envelope is triggered only once at the beginning of each note.

The next six bytes (2 to 7) define the frequency envelope, in the form of three pairs of bytes giving a rate of change of pitch value (from -127 to 127, in 2's complement), and a duration over which this change occurs (from 0 to 255 of the time units defined by byte 1; see above). The rate of change is given in bytes 3, 4 and 5, and the time intervals in bytes 6, 7 and 8; these being for the first, second and third parts of the envelope respectively.

The last six bytes define the amplitude envelope. Bytes 12 and 13 define the amplitude level at two points in the envelope, which we shall call level 1 and level 2. Byte 8 defines the rate of rise to level 1, and byte 9 the rate (from -127 to 127) at which level 2 is reached. Byte 10 defines the rate of fall (from -127 to 0) from when level two is reached to when the sound duration (as given in Osword call 7: Make Sound) expires; and byte 11 describes the rate of fall of amplitude after the sound duration ends (assuming a new note does not cut off the old one).

Osword Call 9: Read Pixel

Passed Values	Returned Values
X, Y coordinates	Pixel value

X and Y coordinates are passed as a low/high pair of bytes, the X value being at byte 0 of the scratchpad and the Y value at byte 2. The logical colour of the specified pixel is returned in byte 4 of the scratchpad.

If an invalid X, Y address is given then the value FF hex is returned.

Osword Call 10: Get Character Software Definition

Passed Values	Returned Values
ASCII character code	Character definition

A character code is passed in byte 0 of the scratchpad. The character representation used for that code is returned as a series of eight bytes (1 to 8). Byte 1 represents the top row from left to right, and so on to byte 8 which represents the bottom row from left to right. A set bit in each byte indicates that the corresponding pixel is in the foreground colour, and a clear bit that the pixel is in the background colour.

Osword Call 11: Read Colour Relationship

Passed Values	Returned Values
Logical colour	Corresponding physical colour

A logical colour code is passed to byte 0 of the scratchpad, and is made modulo the number of colours in the current mode. The corresponding physical colour code is returned in byte 1, these being:

0	Black	8	Flashing Black/White
1	Red	9	Flashing Red/Cyan
2	Green	10	Flashing Green/Magenta
3	Yellow	11	Flashing Yellow/Blue
4	Blue	12	Flashing Blue/Yellow
5	Magenta	13	Flashing Magenta/Green
6	Cyan	14	Flashing Cyan/Red
7	White	15	Flashing White/Black

Bytes 2, 3 and 4 are zeroed; they are reserved for future expansion.

Osword Call 12

This call is reserved for future expansion.

Osword Call 13: Read Graphics Cursor Position

Passed Values	Returned Values
None	Last x, y, current x, y.

The last two x, y positions of the graphics cursor are returned in the scratchpad as 4 low/high byte pairs. The point visited prior to its current position is returned in bytes 0 to 3, and its current position in bytes 4 to 7.

Because the screen has coordinates of 1280 x 1024, but the cursor may only be addressed over a maximum range of 640 x 256, this function rounds values. The x value is divisible by 2, and the y value by 4, with all values rounded down.

4.6 OSBYTE CALL INTERFACE

4.6.0 Accessing Osbyte Calls

Osbyte calls are accessed by User Command 15, which is itself accessed by the Torch Base Command 15. The call number is passed by the Z80 to the 6502, after calling the above commands. Thus to call Osbyte call nn, the following calls are made:

```
tx 0FH, tx 0FH, tx nnH.
```

Parameters for the Osbyte calls are passed in the next two bytes, in the order x, y. Osbyte calls always issue two result bytes to the Z80; these may contain either useful information or (for calls not issuing results) undefined values. Therefore, the Z80 should always then reply:

```
rx xxH, rx yyH.
```

When the call is made, the system sets register A of the 6502 to the call number, register X to the X parameter, and register Y to the Y parameter.

Osbyte calls may also be accessed from either CCCP (the normal Torch command line interpreter) or from the Acorn CLI (used when the computer is in BBC Basic mode, or by Osword call 0) by using the command:

```
*FX n, x, y.
```

When only one parameter has to be passed, the Y parameter may be omitted from the *FX command. Note that this command is of no use where the Osbyte call returns a value, since it may not be accessed by the *FX command.

4.6.1 Osbyte Call List

0	Return Version Number
2	Select Input
3	Select Output
4	Control Cursor Edit
5	Select Printer
6	Set Printer Ignore Character
7	Set Rx Baud Rate
8	Set Tx Baud Rate
9	Set Flash Mark Period
10	Set Flash Space Period
11	Set Auto Repeat Delay
12	Set Auto Repeat Period
15	Flush Buffers
16	Select Analogue to Digital Channels
17	Force Analogue to Digital Conversion
18	Reset Soft Keys
21	Flush Buffer
128	Read Analgue to Digital Channel
132	Get Start of Screen Memory
133	Get Start of Screen Memory for Mode
134	Read Cursor Position
135	Read Character at Cursor Position
138	Write to Buffer
145	Read from Buffer
188	Get Current Analogue to Digital Conversion Channel
189	Get Number of Analogue to Digital Conversion Channels
220	Set Escape Character
221-228	Get/Set Osvariable

4.6.2 Osbyte Call Specifications

Osbyte Call 0: Return Version Number

Passed Values	Returned Values

X= 0
Y= Undefined
	Undefined

or	

X-= 0
Y= Undefined
	X= Version Number
	Y= Undefined

A message is put to the screen, giving the version of the Acorn M.O.S. (Machine Operating System) installed. If the X parameter is non zero on entry, then the version number is given in the X parameter on leaving. It is passed as two hex digits, equivalent to the corresponding decimal values; e.g. 25 hex represents version 2.5, 3A hex represents version 3.10.

Osbyte Call 2: Select Input

Passed Values	Returned Values
X= Console input code	Undefined
Y= Undefined	

The X parameter indicates which device should be treated as console input. The following selections are available, all others causing undefined actions:

0: Read keyboard, and lose RS423 input. (The RS 423 buffer is unaffected)

1: Lose keyboard input; if RS 423 buffer is empty, read from RS 423 input; otherwise read from buffer, and buffer RS 423 input.

2: Read keyboard; add RS 423 input to buffer.

There is no useful value returned.

Osbyte Call 3: Select Output

Passed Values	Returned Values
X= Console output code	Undefined
Y= Undefined	

The X parameter indicates which device should be treated as console output. The following selections are available, all others causing undefined actions:

Screen output	0
Printer output	Not yet defined
RS423 output	3
Special output	Not yet defined

There is no useful value returned.

Osbyte Call 4: Control Cursor Edit

Passed Values	Returned Values

X= Control Code
Y= Undefined
	Undefined

The effect of the cursor controls is set by the control code given. The following values have the following effects, with all other values causing an undefined action:

0 Enables cursor editing

1 Disables cursor editing. The cursor control keys will issue the following character codes:

Copy	87 hex
Left	88 hex
Right	89 hex
Down	8A hex
Up	8B hex

2 Makes the cursor control keys act as extra soft keys, with the following key numbers:

Copy	11
Left	12
Right	13
Down	14
Up	15

There is no useful value returned.

Osbyte Call 5: Select Printer

Passed Values	Returned Values
X= Printer code	Undefined
Y= Undefined	

The printer output specified by the X parameter is used for all printing until the next hard reset. The printer codes are:

0	Sink (i.e. not printed)
1	Centronics Port
2	RS 423 Port

There is no useful value returned.

Osbyte Call 6: Set Printer Ignore Character

Passed Values	Returned Values
X= Character	Undefined
Y= Undefined	

The character code passed as the X parameter is not sent to the printer stream either when using Torch VDU control codes, or when reflecting output to the printer by pressing control-P. It has no effect on any CPN list functions. This is particularly useful if, say, your printer generates a <linefeed> after every <carriage return> it receives.

Osbyte Call 7: Set Rx Baud Rate

Passed Values	Returned Values
X= Rate code	Undefined
Y= Undefined	

The Rx Baud rate for the RS 423 Port is set to the value given by the X parameter. Valid codes are:

1	75 Baud
2	150 Baud
3	300 Baud
4	1200 Baud
5	2400 Baud
6	4800 Baud
7	9600 Baud
8	19200 Baud

All other values cause an undefined action. There is no useful value returned.

Osbyte Call 8: Set Tx Baud Rate

Passed Values	Returned Values
X= Rate code	Undefined
Y= Undefined	

The Tx Baud rate for the RS 423 Port is set to the value given by the X parameter. Valid codes are:

1	75 Baud
2	150 Baud
3	300 Baud
4	1200 Baud
5	2400 Baud
6	4800 Baud
7	9600 Baud
8	19200 Baud

All other values cause an undefined action. There is no useful value returned.

Osbyte Call 9: Set Flash Mark Period

Passed Values	Returned Values

X= Mark period
Y= Undefined
	Undefined

Physical Colour Codes 8 to 15 produce a screen of flashing colours, these being (for physical colour N) the physical colours N-8 and 8-N+7. This call sets the time (in centiseconds) spent in colour N-8 (the first one given in lists of colour codes) to the value of the X parameter.

There is no useful value returned.

Osbyte Call 10: Set Flash Space Period

Passed Values	Returned Values

X= Space period
Y= Undefined
	Undefined

Physical Colour Codes 8 to 15 produce a screen of flashing colours, these being (for physical colour N) the physical colours N-8 and 8-N+7. This call sets the time (in centiseconds) spent in colour 8-N+7 (the second one given in lists of colour codes) to the value of the X parameter.

There is no useful value returned.

Osbyte Call 11: Set Auto Repeat Delay

Passed Values	Returned Values

X= Delay period
Y= Undefined
	Undefined

The delay before a key auto repeats while being depressed is set to the value (in centiseconds) given by the X parameter. A value of zero disables the auto repeat facility.

There is no useful value returned.

Osbyte Call 12: Set Auto Repeat Period

Passed Values	Returned Values
X= Period time Y+ Undefined	Undefined

The period with which a key auto repeats while being depressed is set to the value (in centiseconds) given by the X parameter. A value of zero resets the auto repeat delay and period to their normal default values.

There is no useful value returned.

Osbyte Call 15: Flush Buffers

Passed Values	Returned Values

X= Buffer Code
Y= Undefined
	Undefined

The buffers given by parameter X are flushed. Valid codes are:

0 All buffers
1 Currently selected input buffer.

Any other value has an undefined effect. Values from 2 to 127 are reserved for future expansion: from 128 upwards is available for user applications.

There is no useful value returned.

Osbyte Call 16: Select Analogue to Digital Channels

Passed Values	Returned Values

X= Number of channels
Y= Undefined
	Undefined

On entry, the X parameter specifies the number of channels on which analogue to digital sampling is to occur. If a value of zero is specified, then sampling is suppressed; otherwise the given number of channels are activated (to a maximum of 4).

There is no useful value returned.

Osbyte Call 17: Force Analogue to Digital Conversion

Passed Values	Returned Values

X= Channel number
Y= Undefined
	Undefined

The specified channel (from 1 to 4) has an analogue to digital conversion forced on it. Osbyte Call 128 (Read Analogue to Digital Conversion) may be used to test when a value is ready, by passing it an X parameter of zero. If conversion is incomplete, then the returned Y parameter will be zero. When conversion is completed, the channel number is returned in the Y parameter, and the value may then be read.

There is no useful value returned by this call.

Osbyte Call 18: Reset Soft Keys

Passed Values	Returned Values
Undefined	Undefined

All the soft keys (including 0 to 3, which are normally preset) are redefined to produce a null string (no character codes).

There is no useful value returned.

Osbyte Call 21: Flush Buffer

Passed Values	Returned Values
X= Buffer code	Undefined

The buffer specified by the buffer code is flushed. Valid buffer codes are:

0	Keyboard	5	Sound channel 1
1	RS 423 input	6	Sound channel 2
2	RS 423 output	7	Sound channel 3
3	Printer	8	Speech
4	Sound channel 0		

There is no useful value returned.

Osbyte Call 128: Read Analogue to Digital Channel

Passed Values	Returned Values
X= Channel number	Channel Value
Y= Undefined	
or	
X= 0	X= Fire button status
Y= Undefined	Y= Last channel converted
or	
X= Buffer code	X= Full / empty slots
Y= Undefined	Y= Undefined

On entry, the X parameter may either have the value of zero, or a valid channel number (1 to 4), or a buffer code (-1 to -9). If it is a channel number, then the current value of that channel is returned as a low/high pair in the X and Y parameters respectively.

If X is zero, then the status of the fire buttons is returned in bits 0 and 1 of the X value; a set bit shows the button was depressed, and a clear bit that it was not depressed. Bits 2 to 7 are undefined. The Y parameter returns the number of the channel to last be converted (in the range 0 to 4).

If X is negative, then information is returned in the X parameter giving space allocation in the buffer. For output buffers, the number of empty slots is given; for input buffers, the number of characters present is returned. Valid buffer codes are:

-1	Keyboard	-6	Sound channel 1
-2	RS 423 input	-7	Sound channel 2
-3	RS 423 output	-8	Sound channel 3
-4	Printer	-9	Speech
-5	Sound channel 0		

Osbyte Call 132: Get Start of Screen Memory

Passed Values	Returned Values
Undefined	Start of screen memory

The start of screen memory for the current screen mode is returned as a low/high byte pair in parameters X and Y respectively. (Top of screen memory is always 7FFF hex.)

Osbyte Call 133: Get Start of Screen Memory for Mode

Passed Values	Returned Values
X= Mode	Start of screen memory
Y= Undefined	

The start of screen memory for the mode given by parameter X is returned as a low/high byte pair in parameters X and Y respectively. (Top of screen memory is always 7FFF hex.) If the X parameter does not have a value from 0 to 7, then the action of this call is undefined.

Osbyte Call 134: Read Text Cursor Position

Passed Values	Returned Values
Undefined	X= X coordinate
	Y= Y coordinate

The X and Y parameters returned give the X and Y coordinates respectively of the current text cursor position on the screen. The coordinates are given relative to the current window of text (see Section 4.8 on Character Output), with the top left corner specified as 0,0.

Osbyte Call 135: Read Character at Cursor Position

Passed Values	Returned Values
Undefined	X= Character code
	Y= Undefined

The charater code of the character at the current text cursor position is returned. If the character is one not recognised by the Acorn M.O.S, then a value of zero is returned.

Osbyte Call 138: Write To Buffer

Passed Values	Returned Values
X= Buffer code	Undefined
Y= Character code	

The character code given as the Y parameter is added to the specified buffer. Valid buffer codes are:

0	Keyboard	5	Sound channel 1
1	RS 423 input	6	Sound channel 2
2	RS 423 output	7	Sound channel 3
3	Printer	8	Speech
4	Sound channel 0		

Note that there is no way provided in this call to determine if the buffer is full; Osbyte Call 128 may be useful for this purpose.

There is no useful value returned.

Osbyte Call 145: Read From Buffer

Passed Values	Returned Values
X= Buffer code	X= Undefined
Y= Undefined	Y= Character code

The next character is read from the specified buffer, and returned as the Y parameter. Valid buffer codes are:

0	Keyboard	5	Sound channel 1
1	RS 423 input	6	Sound channel 2
2	RS 423 output	7	Sound channel 3
3	Printer	8	Speech
4	Sound channel 0		

Note that there is no way of determining when a buffer is empty with this function; Osbyte Call 128 may be of use for this purpose.

Osbyte Call 188: Get Current Analogue to Digital Channel

Passed Values	Returned Values

Undefined	X= Current channel
	Y= Undefined

The currently selected analogue to digital channel is returned as the X parameter.

Osbyte Call 189: Get Number of Analogue to Digital Channels

Passed Values	Returned Values

Undefined	X= Number of channels
	Y= Undefined

The number of analogue to digital channels currently selected by Osbyte call 16 (Select Analogue to Digital Channels) is returned as the X parameter. The value lies in the range Ø to 4.

Osbyte Call 220: Set Escape Character

Passed Values	Returned Values
X= Character code	Undefined
Y= Undefined	

The character indicated by the X parameter is redefined to produce the character code 1B hex (escape) and to generate the escape event. The escape key will also still have this effect.

If any characters had been previously set using this call, they are reset to produce their normal character codes.

Osbyte Calls 221-228: Get/Set Osvariable

Passed Values	Returned Values

See below	X= Old value of Osvariable
	Y= Undefined

There are 8 Osvariables in the system which are used to determine the handling of output codes 80 to FF hex. Each Osvariable is controlled by one Osbyte call, and refers to a block of 16 codes, as follows:

221: C0 - CF	225: 80 - 8F
222: D0 - DF	226: 90 - 9F
223: E0 - EF	227: A0 - AF
224: F0 - FF	228: B0 - BF

Each Osvariable has the following effects on its group of output codes, depending on its value:

0: The group of codes is disabled.

1: The code is produced unchanged (the normal default)

Others: The code produced is the value of the Osvariable, plus the value of the bottom nibble (four bits) of the input code.

The Osvariable is altered as follows by this call:

Osvariable := (Osvariable AND Y parameter) XOR X parameter

and the old value of the Osvariable is returned as the X parameter. There is no useful value returned as the Y parameter.

An Osvariable is best written to by passing the new value as the X parameter, and Y as zero. It may be read without alteration by passing X as 00 hex, and Y as FF hex.

4.7 ACORN M.O.S. INTERFACE

Interfaces are provided to some of the commands in the Acorn Machine Operating System. This section should be skipped by those not familiar with the Acorn M.O.S, since all these features are documented elsewhere. This section merely cross-references the interface.

WRCH This is provided by Torch Base Command 21, i.e.

```
tx 15H, tx ch,
```

where ch is the character to be printed.

Osword See the section 'Osword Call Interface' for full details of this interface.

Osbyte See the section 'Osbyte Call Interface' for full details of this interface.

Oscli This command is interfaced by Osword call 0; see the section 'Osword Call Interface' for full details.

Scratchpad See the section 'Osword Call Interface' for full details of this interface.

4.8 CHARACTER OUTPUT

4.8.0 Character Output

This section details the effect of passing control characters to the screen. This may be done either via CPN, or the BIOS vector, or the Torch Base Command Interface, or from the CCCP by the VDU command. The control characters generally produce effects accessible from Torch VDU control codes; where this is so, the Torch SUPERVDU program should be used.

From CPN, control characters may be output either by function 4 (Raw Screen Output) or by function 6 (Direct Console I/O). Two other functions (2: Screen Output and 9: Print String) are available for output to the screen, but these trap control characters and interpret them, as described in their specification.

From the BIOS vector, a character is sent by placing it in Z80 register C, and making a call to location FFD8 hex, e.g:

```
     LD         C,         1B
     CALL       0FFD8
```

would send (escape) to the screen without interpretation.

From the Torch Base Command Interface, a character may be sent direct to the screen by using command 21 (see specification in section 'Torch Base Command Interface').

From CCCP, characters may be output direct to the screen by the VDU command. The values following the command are sent direct to the screen, e.g:

```
     VDU 3, 4, 0, 10, 27
```

would send 3, 4, 0, 10, and 27 in that order to the screen.

Note that different control characters will call for varying numbers of parameters. All parameters called for should be sent, even if they are irrelevant, or have a null value.

4.8.1 Output Code List

The list below gives the following information: 'number' gives the decimal ASCII character code involved, 'hex' gives the hex value of the ASCII character code, 'ctrl' gives the ASCII character which, when pressed with control held down, will produce the code given above, 'bytes' give the number of bytes of parameters needed, and 'function' gives the function name.

Specifications of each command follow this list.

Number	Hex	Ctrl	Bytes	Function
0	00	@	0	<null>
1	01	A	1	Send Character to Printer
2	02	B	0	Enable Printer
3	03	C	0	Disable Printer
4	04	D	0	Separate Cursors
5	05	E	0	Join Cursors
6	06	F	0	Enable VDU Driver
7	07	G	0	Ring Bell
8	08	H	0	Cursor Left
9	09	I	0	Cursor Right
10	0A	J	0	Cursor Down
11	0B	K	0	Cursor Up
12	0C	L	0	Clear Text Area
13	0D	M	0	Move Cursor to Start of Line
14	0E	N	0	Page Mode On
15	0F	O	0	Page Mode Off
16	10	P	0	Clear Graphics Area
17	11	Q	1	Define Text Colour
18	12	R	2	Define Graphics Colour
19	13	S	5	Define Colour Relationship
20	14	T	0	Reset Colour Relationships
21	15	U	0	Disable VDU Driver
22	16	V	1	Select Mode
23	17	W	9	Define Character
24	18	X	8	Define Graphics Window
25	19	Y	5	Plot
26	1A	Z	0	Reset Windows
27	1B	[	0	<escape>
28	1C	\	4	Define Text Window
29	1D	]	4	Define Graphics Origin
30	1E	^	0	Home Text Cursor
31	1F	_	2	Position Text Cursor
127	7F		0	Delete Character

4.8.2 Output Code Specifications

Output Code 0: (null)

This code has no effect on the screen display.

Output Code 1: Send Character to Printer

Passed Values: ASCII Character

The specified character is sent to the printer only, and is therefore not displayed on the screen.

Output Code 2: Enable Printer

Passed Values: None

All characters that are sent to the VDU driver are also sent to the printer. This continues until control character 03 hex is sent to the VDU driver (see below).

Output Code 3: Disable Printer

Passed Values: None

Characters that are sent to the VDU driver are not sent to the printer after this code has been issued, until a code of either 1 or 2 is sent to the VDU driver (see above).

Output Code 4: Separate Cursors

Passed Values: None

The graphics and text cursors are made independent in operation. Text may only be written to the text area, using the text cursor.

This is the normal default state.

Output Code 5: Join Cursors

Passed Values: None

This code causes the text cursor and graphics cursor to be dependent. The two cursors become one, at the screen position of the graphics cursor. This cursor may be moved using output code 25 (Plot) to any position on the graphics area, and text written there. As a result, all scrolling is disabled.

Output Code 6: Enable VDU Driver

Passed Values: None

This code causes all characters to be sent to the VDU driver, usually after the use of output code 21 (Disable VDU Driver).

Output Code 7: Ring Bell

Passed Values: None

A short 'beep' sound is added to the sound queue. It is also sent to the printer.

Output Code 8: Cursor Left

Passed Values: None

This character (backspace) moves the text cursor back one character; if it reaches the start of a line, then it is moved onto the previous line, and if it reaches the start of the text screen, then the text is scrolled down one line. The code has no effect if the start of text is reached.

Output Code 9: Cursor Right

Passed Values: None

This character (tab) moves the text cursor forward one character; if it reaches the end of a line, then it is moved onto the next line, and if it reaches the end of the text screen, then the text is scrolled up one line. The code has no effect if the end of text is reached.

Output Code 10: Cursor Down

Passed Values: None

This character (linefeed) moves the text cursor down one character line; if it reaches the bottom of the text screen, then the text is scrolled up one line. The code has no effect if the last line of text is reached.

Output Code 11: Cursor Up

Passed Values: None

The text cursor is moved up one character line; if it reaches the top of the text screen, then the text is scrolled down one line. The code has no effect if the first line of text is reached.

Output Code 12: Clear Text Area

Passed Values: None

The current text area (by default the whole screen) is cleared and set to the current text background colour (logical colour 7 modulo number of colours in mode). The text cursor is homed to the top left corner of the text area.

Output Code 13: Move Cursor to Start of Line

Passed Values: None

This character (carriage return) moves the text cursor to the left hand edge of the current line. It remains in the text area (which defaults to the whole screen).

Output Code 14: Page Mode On

Passed Values: None

This switches on page mode. Whenever scrolling is to be attempted, the shift key is scanned; if it is depressed, then scrolling takes place; otherwise, scrolling waits until it is depressed.

Output Code 15: Page Mode Off

Passed Values: None

This code sets page mode off, and scrolling takes place continually.

Output Code 16: Clear Graphics Area

Passed Values: None

The graphics area of the screen is cleared and set to the current graphics background colour. The graphics cursor is moved to the bottom left hand corner of the screen.

Output Code 17: Define Text Colour

Passed Values: Colour code

The text foreground and background colours may be set using this code. The colour code, modulo the number of colours available in the current mode, gives a logical colour. If the initial value was less than 128, then the foreground is set to the resultant logical colour; otherwise, it is the background which is changed.

For default logical to physical colour code mappings, see Code 20 (Reset Colour Relationships).

Output Code 18: Define Graphics Colour

Passed Values: Colour handling, colour code

The second byte passed after this code is used in the same way as in output code 17 (Define Text Colour), but as modified by the first byte. The first byte has the following effects:

0	Plot specified colour
1	OR specified colour with that already there
2	AND specified colour with that already there
3	Exclusive OR specified colour with that already there
4	Plot inverse of colour already there.

If the Graphics area is empty, then the colour already there will be the graphics background colour.

Output Code 19: Define Colour Relationship

Passed Values: logical colour code, physical colour code, 0, 0, 0

The given logical colour code, modulo the number of colours in the current mode, is redefined for the current mode of screen, according to the physical colour code given (see output code 17: Define Text Colour). All colour relationships apply only to the current mode, and are cleared when the mode is changed.

It should be noted that redefining logical colour 7 will always set the foreground colour, and redefining logical colour 0 will select the background colour.

Output Code 20: Reset Colour Relationships

Passed Values: None

The default text and graphics foreground and background colours are set, and the normal default logical to physical colour relationship is set up. These are:

Two colour modes

0 = Black
1 = White

Four colour modes

0 = Black
1 = Red
2 = Yellow
3 = White

Sixteen colour modes

0 = Black
1 = Red
2 = Green
3 = Yellow
4 = Blue
5 = Magenta
6 = Cyan
7 = White
8 = Flashing Black/White
9 = Flashing Red/Cyan
10 = Flashing Green/Magenta
11 = Flashing Yellow/Blue
12 = Flashing Blue/Yellow
13 = Flashing Magenta/Green
14 = Flashing Cyan/Red
15 = Flashing White/Black

Output Code 21: Disable VDU Drivers

Passed Values: None

This stops any of the output codes affecting the screen, except for output code 6 (Enable VDU Drivers).

Output Code 22: Select Mode

Passed Values: Mode

The mode given, modulo 8, is selected. Default logical to physical colour relationships are restored, the screen is cleared, and the cursor homed to the top left of the screen.

Available modes are:

0: 2 colours; 80 columns x 32 lines text; 640 x 256 graphics.

1: 4 colours; 40 columns x 32 lines text; 320 x 256 graphics.

2: 16 colours; 20 columns x 32 lines text; 160 x 256 graphics.

3: 2 colours; 80 columns x 25 lines text; no graphics.

4: 2 colours; 40 columns x 32 lines text; 302 x 256 graphics.

5: 4 colours; 20 columns x 32 lines text; 160 x 256 graphics.

6: 2 colours; 40 columns x 25 lines text; no graphics.

7: Teletext display mode; 40 columns x 25 lines. The character set is different to that normally used on the Torch, and characters may not be software defined.

Output Code 23: Define Character

Passed Values: Character code, row representations.

The ASCII character code specified is defined to produce the character given in the following 8 bytes. If a code from 0 to 31, or 127 is given then the code has no effect. Otherwise, each following byte represents a row of the character, with a set bit indicating a pixel of the foreground colour, and a clear bit indicating a pixel of the background colour. The first byte is the top row, the eighth byte is the bottom row; all rows read from left to right.

Output Code 24: Define Graphics Window

Passed Values: Left x, bottom y, right x, top y.

A graphics window is set up, and the graphics cursor homed in it. The graphics cursor may not be moved outside the window, nor may any graphics operations take place outside it.

The window is specified by four low/high byte pairs, specifying (respectively) the left hand edge x coordinate, the bottom edge y coordinate, the right hand edge x coordinate, and the top edge y coordinate.

Output Code 25: Plot

Passed Values: Plot code, x coordinate, y coordinate.

Plot is used to draw points, lines and triangles to the screen, according to the plot code given. These are given in the list below:

0 Move relative to last point

1 Draw line relative to last point in graphics foreground colour

2 Draw line relative to last point in logical inverse colour

3 Draw line relative to last point in graphics background colour

4 Move to absolute position

5 Draw line from last point to absolute position in graphics foreground colour

6 Draw line from last point to absolute position in logical inverse colour

7 Draw line from last point to absolute position in graphics background colour.

8-15 As 0-7, but with the last pixel on the line not filled.

16-23 As 0-7, but with a dotted line instead of a solid one.

24-31 As 0-7, but with a dotted line, and with the last pixel on the line not filled.

64-71 As 0-7, but only the last pixel on any line is filled.

80-87 As 0-7, but plot and fill a triangle. The last two points visited are joined with the specified point to form a triangle, and it is filled.

All other values are reserved for future expansion.

The x and y coordinates are given as a low/high byte pair.

'Relative to last point' means moving by the given x, y coordinates from the last point visited. An 'absolute position' is one given as coordinates on the screen, which is 1280 (0-1279) points wide, and 1024 (0-1023) points high, with its origin at the bottom left. Note, however, that the graphics origin may be moved using output code 29 (Define Graphics Origin).

The logical inverse to a colour is (highest logical colour code for current mode) - (logical colour code); e.g. for a four colour mode:

logical	inverse
0	3
1	2
2	1
3	0

Output Code 26: Reset Windows

Passed Values: None

The text and graphics areas are restored to the normal default of the whole screen, and the graphics origin is set to the bottom left of the screen.

Output Code 27: (escape)

When this character is output from a program, the Torch VDU controls are invoked; see section 3, 'Torch VDU Controls'.

Output Code 28: Define Text Window

Passed Values: Left x, bottom y, right x, top y.

This defines the text area, outside which the text cursor may not be moved (and hence no text may be written). If the text cursor was not in the new text area, it is moved to the top left corner; otherwise, it remains where it was.

The coordinates are given in terms of character cells, the numbers on each axis being dependent on the current mode. The x axis is numbered from 0 to 19, 39 or 79; the y axis from 0 to 24 or 31. The origin is the top left hand corner of the screen. The bytes passed specify respectively the left hand edge x cell, the bottom edge y cell, the right hand edge x cell, and the top edge y cell.

Output Code 29: Define Graphics Origin

Passed Values: x coordinate, y coordinate.

The graphics origin is defined relative to the default origin of 0, 0 at the bottom left of the screen. The coordinate of the new origin is given as two low/high byte pairs.

Output Code 30: Home Text Cursor

Passed Values: None

The text cursor is homed to the top left of the text area.

Output Code 31: Position Text Cursor

Passed Values: x coordinate, y coordinate.

The text cursor is moved to the specified coordinate inside the text area, the position being given relative to the origin of the text area. The coordinates are given as two low/high byte pairs.

Output Code 127: Delete Character

Passed Values: None

The text cursor is moved back one space, and the character cell at that position is set to the text background colour. If the cursor is at the start of a line, it is moved to the end of the previous line; if the text area is at the start of the screen, then the screen is scrolled.

4.9 '*' COMMANDS

The Torch CCCP passes all commands commencing with a star to the Acorn command line interpreter. A large proportion of these commands are used for the B.B.C. microcomputer cassette tape interface, and are hence not documented here. The more important commands are given below:

*FX

The *FX command may be used to access Osbyte calls (see section 4.6: Osbyte Call Interface). Those Osbyte calls that issue results may not be usefully implemented with this call.

*BASIC

The *BASIC command makes the Torch run in Basic, as a B.B.C. microcomputer. Full documentation of this facility is available from your dealer in the 'B.B.C. Microcomputer System User Guide'. To return the computer to normal operation, the command '*CPN' should be used, or the reset button (at the back of the Torch) may be pressed while holding down <control>.

*CPN

The *CPN command returns control to the CPN operating system from B.B.C. Basic, after the *BASIC command has been used.

*KEY

The *KEY command is used to program the blue function keys along the top of the keyboard. These are numbered from 0 to 13, but are unmarked on the keyboard. Whenever hit, they will produce the string that has been programmed into them.

They are programmed by typing:

```
*KEY ( space(s) ) (number) ( space(s) ) (string) (return)
```

to a amaximum of 39 characters.

(number) is the number of the key to be programmed, and (string) is the string which is produced when the key is typed.

Where leading spaces are to be used, the string must be enclosed in double quotes (' " '). In this case, any double quotes in the string must be typed twice, e.g:

```
*KEY 4 "    ""Hello"", he said."
```

would, on typing key 4, produce four leading spaces and then:

```
"Hello", he said.
```

If needed, control characters may be programmed. An example is given where a carriage return is wanted (control-M):

```
*KEY 4 F 4 |MB 6|M
```

This would produce:

```
F 4 (return)B 6(return)
```

See section 4.8 (Character Output) for the effects of control codes on the screen.

If key 10 is programmed, then the text in it is output to screen (and any relevant action taken) whenever the reset button on the back of the Torch is pressed. This is useful for setting up your machine in a manner you like; e.g. to set display colours.

4.10 EXTERNAL INTERFACES

Analogue to Digital Interface

The full specification of this interface has yet to be defined. See Osbyte calls 16, 17, 128, 188 and 189 for currently available information.

Econet Interface

To be defined.

Light Pen Interface

To be defined.

TORCH PROGRAMMERS' GUIDE: SECTION 5

MISCELLANEOUS INFORMATION

5.0

CONTENTS

5.1 CCCP COMMANDS

5.1.0 Command

The CCCP command 'Command' (which can be abbreviated to 'C') is used to present the contents of a file as a series of commands to the Torch. Comprehensive facilities are available for argument substitution.

The command takes the form:

```
C (filename) (arguments)     or     COMMAND (filename) (arguments)
```

If no extension is given to the filename, then a default of '.SUB' is assumed. The filename is used as the input stream to the Torch, and any dollar signs in it are treated as parameter flags. Where '$0' appears, the name of the command file will be substituted. '$1' to '$9' result in the corresponding argument being substituted in the text of the command file. A literal '$' may be produced by the combination '$$'.

The argument list consists of groups of characters, separated by spaces and terminated by the line's end. Backslash may be used to insert special characters into the arguments:

```
\n    :newline
\s    :space
\0    :terminate argument (useful for null argument)
```

5.1.1 Input

This command is used to type in a file. It has the syntax:

```
INPUT (filename) (return)
(file)
(end)
```

Both the command line, and the file itself, are input using CPN Function 10 (Read Keyboard Buffer). For details of the editing facilities available, see the specification of this function (section 2.5.2). Input is ended by pressing the (end) key.

5.1.2 Protect

This is used to protect filenames in varying ways. The syntax is:

```
PROTECT <filename> [<options>] <return>
```

The options may be a single string, or may be separated by commas and spaces for clarity. Available options are:

```
either    s: System use only
or        u: Current user only

and one of:
          e: execute/only
          r: read/only
          w: read/write
```

5.1.3 View

View sends a file direct to the screen, without the normal interpretation of control characters. For the effect this has on character output, see section 4.8 (Character Output). This command is most useful for inspecting files of graphics information.

The syntax is:

```
VIEW <filename> <return>
```

5.1.3 [System

This has the syntax:

[System (number)(return)

It has exactly the same effect as User Command 10 (Select Debug State) . The state is affected by each bit in the passed number; if a bit is set it has the following effect (bit 7 being the high bit):

Bit	Effect
0	The top bit of the numeric key pad is set
1	The keys 1 - 9 move the cursor about the screen, horizontally, vertically or diagonally. '0' is defined as 'copy'.
2	Reserved for future expansion
3	Reserved for future expansion
4	Reserved for future expansion
5	When a disc is being used in a read or write operation, the block number being used is printed to the screen, with an 'r'for read or a 'w' for write.
6	On a disc error, there is no hobnailed boot. (This is reserved for test use only).
7	All bytes passed between the 6502 and the Z80 are traced on the screen.

5.2 KEYBOARD CODES

Blue Function Keys

The codes produced by these keys are unaffected by either the shift key or the control key. Codes produced are:

Function 0 (the leftmost function key)	80 hex
through to	
Function 13 (the rightmost unmarked function key)	8D hex
Lower Case	8E hex
Upper Case	8F hex
Move Left	8C hex
Move Past	8B hex
Move Right	8D hex

Deletion Keys

The codes produced by these keys are unaffected by either the shift key or the control key. Codes produced are:

Para	A3 hex
File	A6 hex
Undo	AB hex
Window	A5 hex
Screen	A4 hex
Underline	5F hex
Word	AC hex
Line	A1 hex
Redo	A2 hex
Insert	A7 hex
Begin	A9 hex
End	AA hex
Control	See below

Main Keyboard

The keys on the main keyboard are generally affected by both the shift key and the control key. The table below gives a key's lower case character and code, its shifted character and code, and the codes produced when the key is held with either the control key, or both the shift key and the control key. (All values are in hex).

		(shift)	(shift)		(shift) &
Char	Code	Char	Code	(control)	(control)
Tabulate	09	Tabulate	09	09	09
Return	0D	Return	0D	0D	0D
Line	0D	Line	0D	0D	0D
Escape	1B	Escape	1B	1B	1B
(space)	20	(space)	20	20	20
"'"	27	'"'	22	27	22
'('	28	'{'	7B	28	7B
')'	29	'}'	7D	29	7D
','	2C	'?'	3F	2C	3F
'-'	2D	'+'	2B	2D	2B
'.'	2E	'!'	21	2E	21
'0'	30	'='	3D	30	3D
'1'	31	'*'	2A	31	2A
'2'	32	'\'	5C	32	1C
'3'	33	'/'	2F	33	2F
'4'	34	'@'	40	34	00
'5'	35	(pound)	60	35	60
'6'	36	'$'	24	36	24
'7'	37	'&'	26	37	26
'8'	38	'%'	25	38	25
'9'	39	'#'	23	39	23
';'	3B	':'	3A	3B	3A
'['	5B	'<'	3C	1B	1B
']'	5D	'>'	3E	5D	3E
'^'	5E	'-'	7E	1E	1E
'a'	61	'A'	41	01	01
		through to			
'z'	7A	'Z'	5A	1A	1A
'|'	7C	(acute)	27	7C	27
(exact..)	A0	(exact..)	A0	A0	A0

Capitals: redefines keys a-z to be always shifted.

Numeric Pad

The numeric pad keys are functionally the same as keys 0 to 9 and the keys '-' and '.'; that is, they produce the following codes:

'-'	2D	'+'	2B	2D	2B
'.'	2E	'!'	21	2E	21
'0'	30	'='	3D	30	3D
'1'	31	'*'	2A	31	2A
'2'	32	'\'	5C	32	1C
'3'	33	'/'	2F	33	2F
'4'	34	'@'	40	34	00
'5'	35	(pound)	60	35	60
'6'	36	'$'	24	36	24
'7'	37	'&'	26	37	26
'8'	38	'%'	25	38	25
'9'	39	'#'	23	39	23

TORCH PROGRAMMERS' GUIDE: SECTION 6

INDEX

6.0

CONTENTS

6.1 CONVENTIONS AND ABBREVIATIONS

Throughout this index, the following conventions and abbreviations have been used:

Angle brackets are used to indicate a class of entry, and not a literal entry; e.g.
See also <language names>
means to look up the names of all languages you are interested in; say Fortran, Basic and Pascal.

A character in quotes (e.g. '?') is used to indicate an ASCII character. All such entries precede the main index.

All entries commencing with a number also precede the main index.

(CPN) indicates a CPN function.

(VDU) indicates a Torch VDU control code.

(TB Com) indicates a Torch Base Command.

(TB User) indicates a Torch Base User Command.

(TB Osword) indicates a Torch Base Osword Call.

(TB Osbyte) indicates a Torch Base Osbyte Call.

(TB Output) indicates a Torch Base Output Code.

In general, all (TB...) entries should not be referenced where there is an alternative, since CPN and Torch VDU control codes are the preferred interfaces.

6.2 INDEX

TORCH
COMPUTERS

Programmers' Guide