Torch Programmer's Guide

TORCH
COMPUTERS

© TORCH COMPUTERS LTD
Abberley House, Great Shelford, Cambridge CB2 5LQ

TORCHMAIL Telephone (0223) 840238

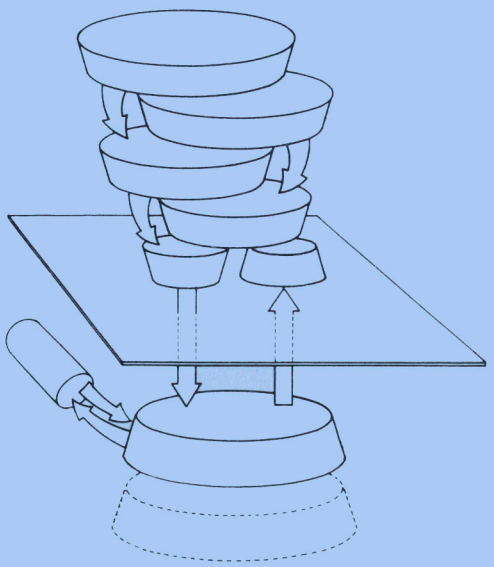
Password: Secret.

Note that, in addition to this Programmers' Guide, there is an important companion volume: the TORCH System Guide. This provides a useful run-up to the Programmers' Guide, covering the complementary topics of built-in commands, files and filenames, the TORCH floppy and hard disc system, printer and other peripheral connections, comparison with and customisation of standard CP/M, preparing command files and the BASIC interpreter mode. In addition, the System Guide explains the use of the utility programs supplied with the TORCH. These are FONT, EXEC, TORCHBUG, RS423, MUSIC, MAPDISC and POKEDISC.

All newcomers to the TORCH should initially refer to the Users' Guide, exhaust the contents of the System Guide, then examine the present manual for help with the 'systems programming' aspects of interfacing the TORCH with Assembler and higher level programming languages.

In addition, the TORCH Reference Card offers a convenient precis of the TORCH System as well as a concise list of CPN functions, VDU codes for direct console output and the ASCII table.

Contents

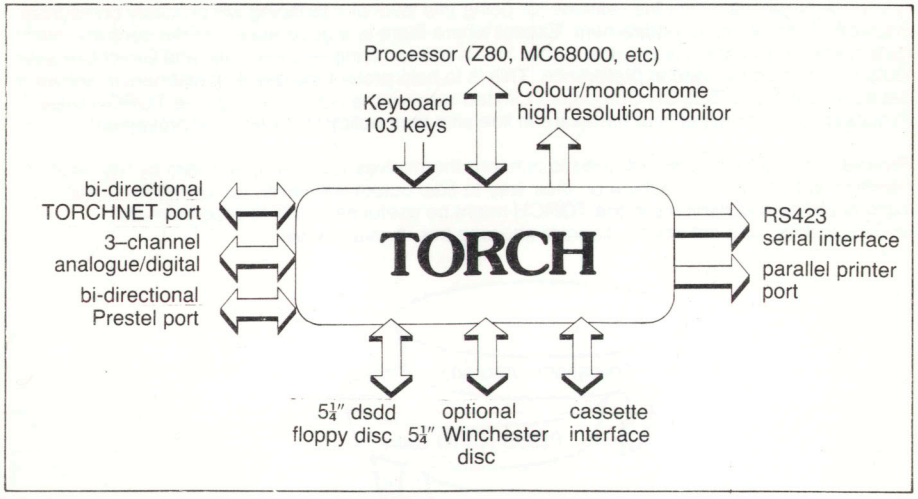


Introduction

Introduction

1.0 System Overview and Contents Guide

The TORCH is a dual processor microcomputer of an advanced and novel design. An 'Applications Processor' (normally a Z80A with 64 KBytes of RAM) is coupled to a 'Base Processor' (at present a 6502A) delegated the task of peripheral handling. A typical TORCH configuration showing the range of ports and I/O devices is given in the diagram below:



TORCH Configuration Schematic

Programmers new to the TORCH need not be daunted by the apparent complexity resulting from the range of levels at which commands may be introduced into the machine. The interface choices give the machine immense power and flexibility and need not cause any extra difficulty.

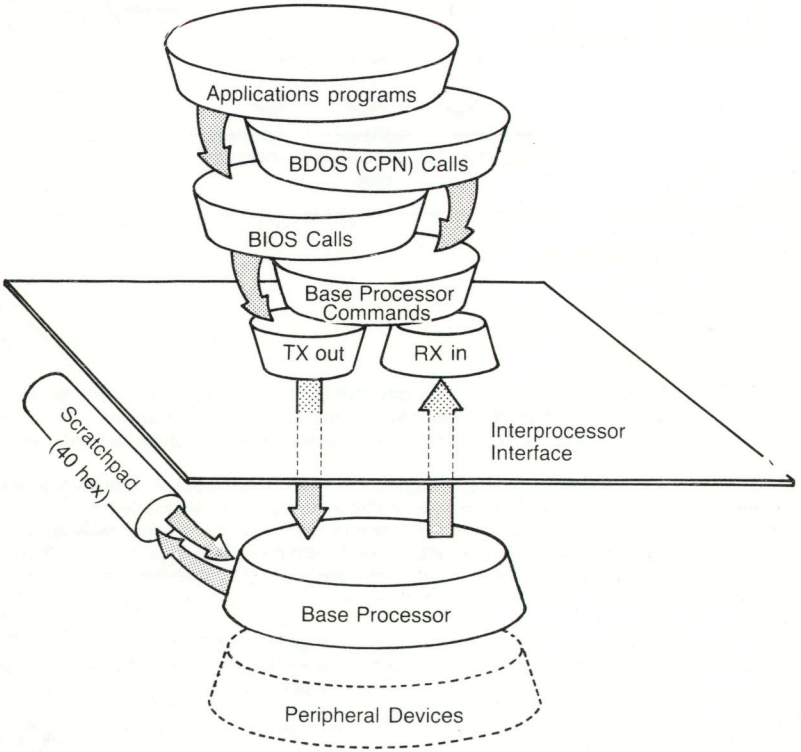
To the user of applications programs or packages, the TORCH behaves in a totally transparent way. Similarly, a programmer using an already configured high level language (and there are many well structured languages to choose from) simply uses the Input/Output facilities within the language itself. Where more detailed control of devices is required, calls to the Operating System (i.e. CPN Calls) may be made in a straightforward way. Programmers familiar with CP/M calls will find little difference between the two systems.

Very often the programmer may simply wish to configure an already existing program for the TORCH. (There is a section in the TORCH Systems Guide entitled 'Customisation of standard CP/M programs for the TORCH' which gives a simple 'recipe' for such an undertaking.) Straightforward character positioning by matching cursor control codes may be all that is necessary to run the program. But for more advanced control of the sophisticated graphics facilities available on the TORCH, the systems programmer is directed straight to Section 6 — 'Direct Console Output'.

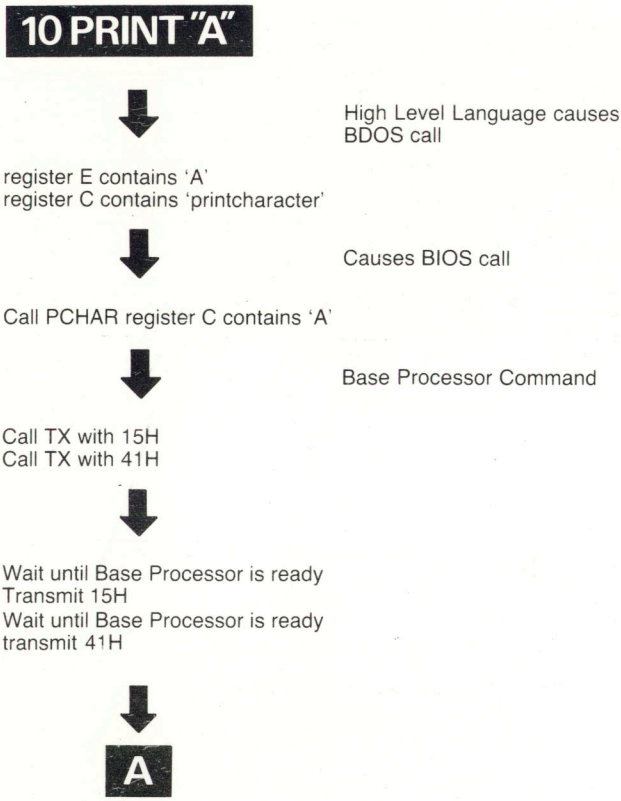
If for some reason control codes for screen output cannot be satisfactorily issued from within the applications program (due to an inadequacy of the programming language used, for example), then the programmer has recourse by loading and using the SUPERVDU program. See Section 7 — 'SUPERVDU functions' — for more details.

Other parts in this manual, such as Section 4 (TORCH Base Processor Calls) and Section 5 (Acorn MOS Interface) detail the lower levels of machine interfacing. In the case of low level commands to the Base Processor, the BBC Microcomputer User Guide provides additional material which the programmer may well find useful or interesting. The BBC Guide is available for purchase separately. The material in Sections 4 and 5 are made available largely for reference purposes, since the reasons for using this level of interfacing will probably come from a specific programming requirement. Except where there is a good reason to the contrary (such as a speed critical application), the higher levels of interfacing — CPN Calls and Direct Console Output — should be used in preference. This is to help protect the users' investment in software development, since TORCH Computers Limited reserve the right to change the TORCH Base Processor or other aspects of hardware in line with their policy of continual improvement.

Programmers generally do not need to concern themselves with the route taken by high level commands through the machine on their way to final output. However, an 'overview' of the various calling mechanisms in the TORCH might be useful as an aid to programmers' understanding, so a schematic diagram showing this is given below:

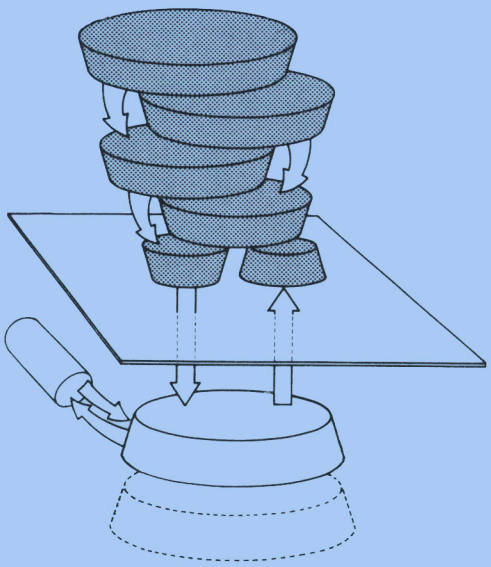


As a further illustration of calling mechanisms, we can follow the course of a simple high level language command (in this case, a simple BASIC 'PRINT' statement) through the machine.



Some applications programs will 'know' the faster route through these series of calls, as was indicated by the 'short-circuiting' arrows on the previous diagram. For instance, WordStar will skip some BDOS calls and call the BIOS vector directly. The experienced programmer can use the same techniques, as long as the warnings about the consequences of possible future changes in hardware specifications given above are heeded.

TORCH
COMPUTERS



CPN Memory and Data Structures

CPN Memory and Data Structures

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 any other interface (such as TORCH Base Commands, User Commands, Osword or Osbyte Calls), CPN commands should always be used.

There are four categories of memory which may be usefully distinguished within the CPN operating system. These are:

i) The TORCH Control Kernel (TCK)

ii) The Basic Disc Operating System (BDOS)

iii) The Cambridge Console Command Processor (CCCP)

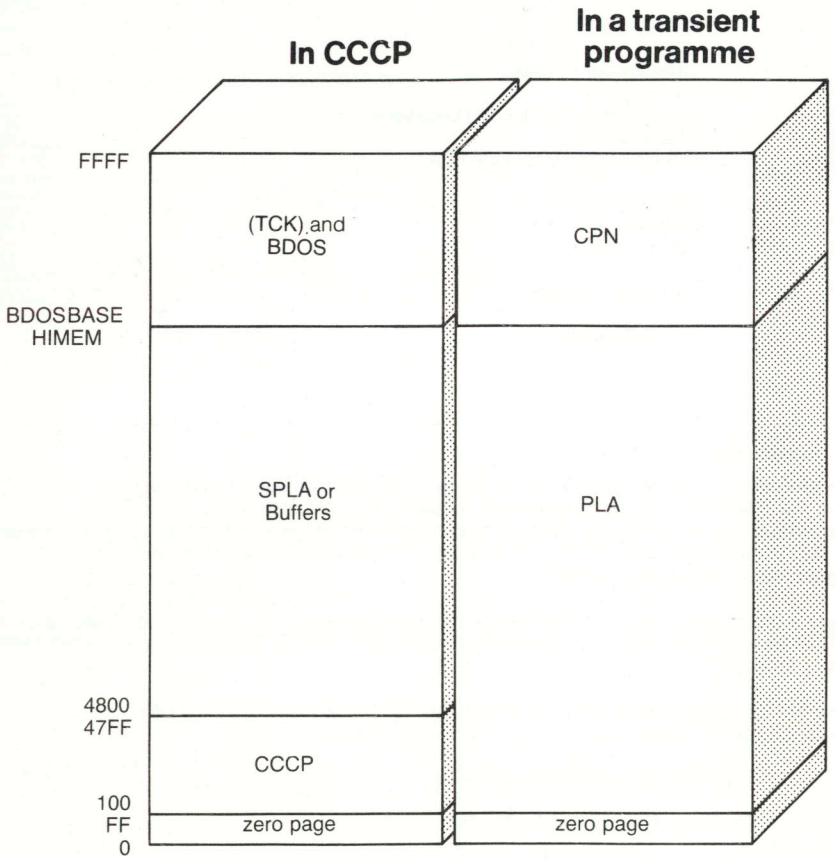
iv) The Program Load Area (PLA)

The TORCH Control Kernel is implemented in Read Only Memory on the Applications Processor board and controls many of the I/O functions. It is usually considered as a single unit along with the BDOS (Basic Disc Operating System) and shares a common entry address.
The PLA is the area of memory used to execute user (i.e. applications) programs and non-resident operating system commands. In CP/M parlance this is known as the 'Transient Program Area', or TPA.

In addition, there is a small area of memory (0000 hex to 0100 hex) below the PLA which is reserved for systems information. This area, the 'zero page', is detailed in section 2.2.1 (Low Memory Locations).

2.2 Memory Layout

2.2.0 Memory Map



The exact memory address of BDOSBASE will vary depending on the CPN version being used. However the system parameters always run at the base of random access memory from 0000 hex to 00FF hex. These locations contain the code (starting at 0000 hex) used to perform a warm start on the system. Each warm start loads and initialises all the CPN and CCCP code before returning control to the CCCP.

This means that to return control to the CCCP commands, all that any user program has to do is jump to location 0000 hex, at which point the system will be reinitialised. Note that the BIOS vector is not reloaded by this type of 'warm boot', nor are locations 0006 and 0007 hex changed. Since this is where the address of BDOSBASE is held, some programs (for example, SUPERVDU.COM) will survive a warm boot and can only be eliminated by a 'CTRL C' operation.

Memory from 0100 hex up to (BDOSBASE — 1) is available for applications programs.

2.2.1 Low Memory (Zero Page) Locations

The low memory locations given below are occupied by the following system parameters:

00–02	Jump to warm boot	set by warm boot
03	IObyte	set to 0 by warm boot
04	Current logged drive	set on cold boot
05–07	Jump to BDOS BASE	normal entry, set on cold boot
08–2F	User RST1 to RST5	these are destroyed on warm boot (used by CCCP but not by CPN — so the user can write code to access them.)
30–32	RST6	Jump to fast CPN, set on cold boot
33–37	Reserved for CPN	
38–3A	RST7	user debug area — set to RET on cold boot
3B–5B	Reserved for CPN	
5C–7C	Main part of default FCB set on entry to PLA	
7D–7F	Random record number of default FCB	
80–FF	Default DMA area	set to command line tail, destroyed by warm boot.

2.2.2 BDOS Entry

Entry to the BDOS is made at location 0005 hex where a jump instruction to BDOSBASE is found. It follows that the address of BDOSBASE may be found at 0006 hex, from which it is possible to calculate the size of available user memory.

2.2.3 Running Programs in the PLA

Once CCCP is loaded into memory, programs may be 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 significant string(s) that could be filenames are used.

As a convenience, the Dynamic Memory Access buffer (DMA buffer; 0080 to 00FF hex by default; see section 2.4) is set to the last part of the command line. The first position of the buffer is set to the total number of characters in the argument, excluding the final <carriage return>. It is followed by the actual characters typed but with lower case letters translated to their upper case equivalents. This is terminated by a carriage return, followed by a section of uninitialised memory. So, in the following example:

```
COPY This to That
```

the DMA buffer would be set to:

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	hex
13		T	H	I	S		T	O		T	H	A	T	<cr>	??	

For commands which are followed by filenames, a File Control Block (FCB) will be set up by the CCCP in the area reserved for default FCB use (i.e. 005C–007F hex). These are used for accessing the files from the BDOS. For further details of how these are established, see Section 2.4 (File Control Blocks).

The command may be one of the standard 'built-in' CCCP 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 a user command, the executable code contained in the file '<command>.COM' will be loaded from disc into the PLA starting at location 0100 hex.

The CCCP hands over 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 instruction 'RET', provided that the stack pointer is saved.

Alternatively, the program may use a jump to 0000 hex or RST 0 to achieve the same effect.

2.3 File Structure

2.3.0 File Names

In CPN a file is referred to by a disc file name. It has three components:

1. A drive select letter, followed by a colon ':',
2. a file name consisting of between one and eight non-blank characters, and
3. a file type consisting of between zero and three non-blank characters (preceded by a full stop '.').

If no drive select letter is used then the current default disc is assumed. Initially this is the logged-in drive, but it may be changed using CPN function 14 (Select Disc).

The second component, the descriptive file name, is used to distinguish between different files of the same type.

The final component of the file name, the optional file type, indicates the nature of the file contents using a locally agreed convention. Some of the more commonly used suffixes are given as examples below.

.@@@	reserved for CPN filing system
.$$$	a temporary or incorrectly saved file
.ABS	absolutely located code
.ASM	an assembly language program
.BAK	a backup file
.BAS	a program in MicroSoft MBASIC language
.BBC	a program in BBC BASIC
.BPL	a BCPL source code file
.C	a C source code file
.CDX	a Cardex data file
.CBL	a Cis COBOL program
.COB	a COBOL80 program
.COM	a program that can be directly run from CCCP
.DAT	a data file
.DES	Software description file
.DOC	Software documentation file
.DRY	a Diary data file
.FNT	Fount (pron. 'font') data
.FOR	a FORTRAN source code program
.H	BCPL and C header files
.HEX	a program in Intel's Hex format
.INT	an intermediate file created by a program
.LET	a file containing the text of a letter
.MAC	a macro assembler source file
.MSC	Music manuscript file
.MSG	TORCHMAIL messages file
.MUS	Music compressed code
.OVL	Overlay section library
.OVR	Overlay section library
.PAS	a Pascal source file
.PHN	telephone and Telex numbers
.PIC	Picture file for Viewing
.PLI	a PL/I program
.PLM	a PL/M program
.PLZ	a PL/Z program
.PRN	a text file for direct printing
.REL	relocatable code (M80 assembler)
.SUB	a text file for use with CPN's Submit program

2.3.1 Disc Layout

Full details of the CPN floppy disc file structure are given in Appendix C of the TORCH System Guide. Included in the Appendix is information on notation, physical, logical and file structures, the directory and allocation map. More esoteric topics, such as L2 and L3 blocks and 'special' discs are also covered.

Briefly, each disc contains a directory of the files on it as well as an area of file data. This format allows a disc to contain a variable number of records dispersed over the data area.

The file itself is stored on the data area of the disc in up to 32K records (numbered from 0 to 32767) 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 so it is only possible to directly access 4 Mbytes. 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. These consist of a representation of the disc file name and some system information. An FCB is 33 bytes long for sequential access of files and 36 bytes long for random access. There is a default FCB area of 36 bytes reserved in memory by CPN starting at 005C hex. 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.

Most CPN functions from 15 upwards use the register pair DE to address an FCB.

The structure of an FCB is as follows:

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	uF	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 (bit 7) used. The high bits of f1-f4 are available for programmer use.

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: is reserved for system use.

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 before opening the file.

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

h1 Reserved for use by the system.

h2 Reserved for use by the system. This byte is set to zero by the system on a call to the 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–uF Contains the user field. These bytes are reserved for system use — but see the final paragraph in this section for the use of these bytes in the default FCB area and with calls having two parameters.

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 with each sequential read.

r0–r2 Contains the current record to use in a random I/O operation. The value is contained in bytes 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 the first 16 bytes of an FCB in the user field of the default FCB.

Note: it is the responsibility of the programmer to clear the cr and ex bytes of the FCB before opening the file.

2.5 The BIOS Vector

High Memory controlled by the Z80 applications processor is taken up by the BIOS vector, which provides a series of jumps to routines useful to the programmer. All the direct BIOS calls are available to the user, except for 'CInstall'. Most of them use a standard calling sequence or protocol for which a description is given following the list.

Note that CPN calls and BIOS vector calls only make use of Z80 registers A F B C D E H and L. The alternate register set (A' B' etc.), IX and IY are unused. CPN calls use their own stack, preserving the main stack pointer for recovery on exit whereas BIOS vector calls use the main stack. The CCCP uses a call to LdFile to execute transient files (i.e. applications programs).

Address	Symbolic Address	Action	Protocol
FFFF	GetVersion	CPN Version No. (byte)	R
FFFC	SecTran	(sector translate)	Null
FFF9	ListStat	get printer status	R
FFF6	WriteDsk	(write disc sector)	Null
FFF3	ReadDsk	(read disc sector)	Null
FFF0	SetDma	set DMA address	see note 1
FFED	SetSec	(select sector)	Null
FFEA	SetTrk	(select track)	Null
FFE7	SelDsk	select disc drive	see note 2
FFE4	Home	(home disc drive)	Null
FFE1	Reader	(get char from reader)	see note 3
FFDE	Punch	(write char to punch)	see note 4
FFDB	ListCh	write char to printer	W
FFD8	PutCh	write char to screen	W
FFD5	GetKey	get char from keyboard	R
FFD2	ConStat	get keyboard status	see note 5
FFCF	WBoot	warm boot	see note 6
FFCC	CBoot	cold boot (^C)	see note 7
		end of CP/M standard BIOS	
		start of CPN BIOS	
FFC9	PutByte	send byte to TUBE	W
FFC6	GetByte	get byte from TUBE	R
FFC3	PutImm	send Imm byte to TUBE	I
FFC0	UsrImm	usr call (Imm byte)	I
FFBD	LdFile	load & execute COM file	see note 8
FFBA	CInstall	(install SUB file)	see note 9
FFB7	TubeStat	get TUBE status	see note 10
FFB4	CCCPCS	(reserved, do not use)	

Protocols:

R

A byte is read into register A; flags are set by AND A.

W

A byte is sent from register C. Register A has the contents of C assigned to it; F, the flag register, is corrupted.

I

'Immediate bytes' are sent from ((stack pointer)). The value on the top of the stack is incremented, to allow a correct return. Register A is assigned the byte sent and F is corrupted.

LdFile

On entry, BC is assigned the FCB of the program to be run and the BIOS vector entry 'WBoot' (pointed to by the contents of memory location 0001) is changed.

If the file is successfully opened, the entire TPA (Transient Program Area) is corrupted before loading the file. This means that 'LdFile' should not be used if memory overlaying techniques are used. On entry to the program, register pair DE contains 0080 hex and HL contain 0. A F B C are undefined.

If the file is not opened, register A is assigned the error code from 'open'. As side effects, DE contains the FCB actually used, BC is unchanged but F and HL are corrupted. Return is made via 'RET'.

The stack pointer points to a 32 byte stack (capable of retaining a maximum of 16 two byte addresses). Underneath these entries and not included in the 32 bytes the value '0' has been pushed. Location '0' contains the address of the 'jump to warm boot' code. Thus 'RET' returns control to CPN in every case.

TubeStat

The register pair AF is set on exit as follows:

A

7	6	5	4	3	2	1	0
0	0	0	0	0	r	0	t

F

S	Z	—	H	—	PV	N	C
?	~ r	—	?	—	?	?	t

where:
r–1 data waiting to be read
r–0 no data waiting to be read

t–1 TUBE clear to send character
t–0 TUBE not clear to send character

These settings may be used by the programmer by testing bits 0 and 2 directly from register A, or by using the instructions

```
JP      Z,a
JR      NC,e
RET     C
```

for example, testing the condition of NZ, Z, NC or C.

Notes:

1. The new DMA address is taken from register pair BC. All registers are left unaltered.

2. The disc drive number is given in register C. Values 0 to 15 correspond to drives A: to P:. Register pair AF is corrupted.

3. Because there is no support for punch and reader devices, 'reading' is achieved by 'GetKey'.

4. As note 3., 'punch' is achieved by 'PutCh'.

5. Register pair BC is corrupted

6. Transient programs stop; 'warm boot' code is contained in the area pointed at by the contents of location 0001.

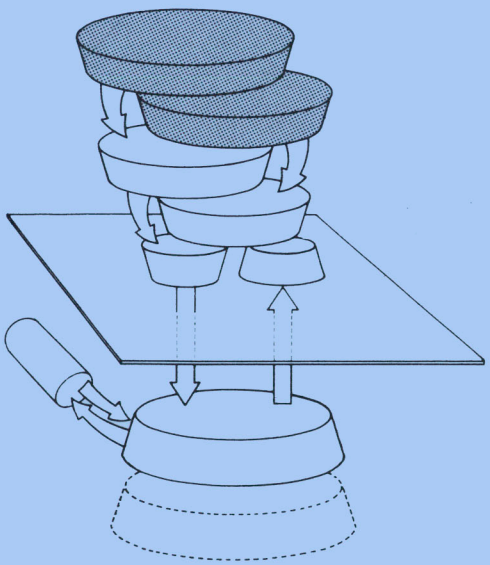
7. A 'cold boot' command (equivalent to '^C') stops execution of transient programs.

8. The protocol is described above, as 'LdFile'.

9. Use of this BIOS call is not recommended since the entry point is provided for CCCP use — the code for installing a SUB file is part of the CCCP.

10. The protocol is described above, as 'TubeStat'.

TORCH
COMPUTERS



CPN Interface Information

CPN Interface Information

3.0 Contents

Where a programmer has a choice between using CPN and using any other interface (such as Torch Base Commands, User Commands, Osword or Osbyte Calls), CPN commands should always be used.

3.1 Operating System Calls

3.2 Accessing CPN Functions

To communicate with the keyboard, the disc operating system and other external peripherals, the user program uses CPN I/O facilities. To access the I/O system a function number must be passed to CPN, usually along 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 while any other values are passed in register E, or the double register DE. Results are returned as follows:

1. in register A if of single byte length, with L = A; or

2. in registers HL if of double byte length, with A = 'Low' and B = 'High' (for historical reasons in the processor design).

Thus with single byte results, only registers A and L are affected; but with double byte results A, B, C, H and L may all be changed.

CPN calling conventions are tabulated in Section 3.4, below. This gives more details of the state of registers after CPN functions have been called.

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     0030 hex                    ;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. 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.

3.3 Function List

Available CPN functions are given in the list below. The calling conventions are given in a summary table (3.4) and then the calls themselves with input parameters and results returned are described in more detail in the section which follows (3.5).

The functions in the list below which are marked with an asterisk are provided for compatibility with CP/M only.

Function	Name
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

Function	Name
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 (Allocation)
28	Write Protect Disc
29	Get Read/Only Vector
30	Set File Attributes
31	Get Address (Disc Parameters)
32	Set/Get User Number
33	Read Random
34	Write Random
35	Compute File Size
36	Set Random Record
37	Reset Drive
40	Write Random With Zero Fill

3.4 CPN Calling Conventions

	All Calls	Call 5	Call 30
All Calls	DE holds argument AF' BC' DE' HL' IX IY are unchanged	(SP)→dw ret addr returns C holds call no.	(SP)→ db call no. (SP)→ dw ret addr returns BC unchanged
Good Call, result in HL*	A = result low F : C clear ZM according to A else ? H = result high L = result low	B = result high C = ?	
Good Call, result in A (if any)	A = result F : C clear ZM according to A else ?	BC unchanged H = 0 L = result	HL unchanged
Bad Call	A = 0 F : C set else ? HL unchanged	BC unchanged	

* For example, CPN Calls 12, 24, 27, 29, 31; CPNet Calls 111, 113, 114, 127

3.5 Call Specifications

Function 0: System Reset

Passed Values	*Returned Values*
None	None

Function zero re-enters the CPN operating system. It has exactly the same effect as jumping to 0000 hex, that is, the disc system is reinitialised.

Function 1: Keyboard Input

Passed Values	*Returned Values*
None	Register A: ASCII character

Function one reads a character from the keyboard to register A. If no character is present in the keyboard buffer then execution is suspended until a character is entered.

Characters are reflected to the screen after the following treatment:

Graphics character as below	Bit 7 is set to 0, then interpreted
Carriage return	Reflected
Line feed	Reflected
New line	Reflected
Tab	Expanded into a column of up to 8 spaces
Escape	Invokes Torch VDU controls (see Section 6)

Other control combinations are echoed as ^<letter>

Function 2: Screen Output

Passed Values	*Returned Values*
Register E: ASCII character	None

Function two 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) above.

Function 3: Raw Keyboard Input

Passed Values	*Returned Values*
None	Register A: ASCII character

Function three stores the next character typed at the keyboard in register A. It differs from function 1 (Keyboard Input) in that there is no interpretation of any of the characters, including control characters.

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 suspends interpretation of control characters. It is available under CPN for specialist programming uses only.

Function 4: Raw Screen Output

Passed Values	*Returned Values*
Register E: ASCII character	None

Function four outputs the character stored in register E to the screen. There is no interpretation of the output character. So, for example, tabs are not expanded into spaces and printer echo is unchecked. See Section 4.8 (Character Output) for further details. As with Function 3, this function is not used for Punch Output, since the TORCH does not support a paper tape punch.

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

Function 5: Printer Output

Passed Values	*Returned Values*
Register E: ASCII character	None

Function five 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) or ASCII character (output)	Register A: ASCII character or status (input) or No value (output)

Function six provides facilities for raw 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 the 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 suspends interpretation of control characters. It is available under CPN for specialist programming uses only.

Function 7: Get I/O Byte

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

Function seven 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

Function eight 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

Function nine 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 interpreted as in function 1 (Keyboard Input).

Function 10: Read Keyboard Buffer

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

Function ten reads a line of keyboard input into a buffer addressed by registers DE.

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

Rubout/Delete/ Control-H	Remove the 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.

Note: The 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

Function eleven 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

Function twelve returns a code in registers HL for the version of CPN installed.

Register H contains the value '0' if the operating system is CPN and '1' if it is MPN that is implemented.

Register L contains a hex representation of the version number of CP/M with which CPN is compatible. So, for example, if L contains 22 hex, the current level of compatibility is with version 2.2 of CP/M.

This function can be usefully incorporated in application software to produce an error message if the installed operating system cannot implement a particular feature.

Function 13: Reset Disc System

Passed Values	*Returned Values*
None	None

Function thirteen resets the access state of all discs to Read/Write (see functions 28 and 29) and sets the default DMA to 0080 hex (see function 26).

Function 14: Select Disc

Passed Values	*Returned Values*
Register E: Disc to select	None

Function fourteen sets the default disc drive 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 fully extended 16 drive system.

The default drive is used whenever an FCB specifies a drive code of 0. Drives A: to P: can be explicitly selected by using drive codes of 1 to 16.

Function 15: Open File

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

Function fifteen opens an already created file in the current user's disc directory. If a file cannot be found for the currently set user, then the search will be continued for a name match under user 0.

BDOS scans the relevant directory for an FCB matching the one addressed by the contents of registers DE in bytes 1–12. A '?' will match any character in the scanned directory. This is a useful when 'wild cards' have been used in the original command. In such a case the first successful match made is used. The system automatically zeroes bytes ex, r1 and r2. To access a file sequentially from the first record byte cr must be explicitly set to zero by the programmer.

If a match is made, then bytes 00 to 12 and byte 15 of the matched FCB in the directory are copied into the user field of the FCB. The u0 byte of the FCB is corrupted and Register A is set to 00 hex. If no match is made, then no alteration is made and register A is set to FF hex. In both cases, the h2 byte of the FCB is cleared.

Note that no file can be accessed before it has been opened.

Function 16: Close File

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

Function sixteen 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 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

Function seventeen examines the file directory for the first occurence of an FCB matching the one 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 is then returned, whether or not the s1 byte is set. On the other hand, if the dr byte is not set to '?', then the first match belonging to 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 both cases, the h2 byte of the FCB is cleared.

Function 18: Search For Next

Passed Values	*Returned Values*
None	Register A: Return Code

Function eighteen finds the next occurence of a file, following a previous call of a Search function. It may be used repetitively, but must be preceded by calling 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

Function nineteen removes all files whose FCB matches those addressed by registers DE. The match is made in the same way as in function 15 (Open File), unless the dr byte is a '?'.

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.

Function 20: Read Sequential

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

Function twenty reads the next record (in sequential order) 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. The byte is incremented on each read. If overflow occurs, 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 an end of file condition 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

Function 21 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). The cr byte 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 existing 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

Function 22 creates a new file and Opens it. The FCB addressed by DE is used to name the new file so it should not already exist in the referenced disc directory. Use of function 19 (Delete File) before this function will ensure that this does not occur (unless the file is explicitly protected). BDOS will initialise the file directory and main memory value as an empty file before creating the file and activating the FCB.

If the operation is successful, then 00 hex is returned in register A. If it is unsuccessful (for example, if no more directory space is available) then FF hex is returned. In both cases the h2 byte of the FCB is cleared.

Function 23: Rename File

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

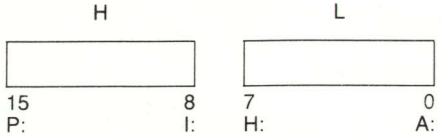
Function 23 renames the file referenced by the FCB with the file name contained in the user fielc of the FCB. As usual, the FCB is addressed by registers DE. The drive code for the user field is ignored.

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.

Function 24: Return Login Vector

Passed Values	*Returned Values*
None	Registers HL: Login vector

Function 24 determines which drives are available by returning a login vector. The login vector value is a 16-bit value returned in registers HL. The least significant bit of L refers to drive A: while the most significant bit of H refers to drive P:.



A zero indicates a drive does not exist; a one indicates that a drive does exist.

Function 25: Return Current Disc

Passed Values	*Returned Values*
None	Register A: Current Disc

Function 25 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

Function 26 is used to set the Direct Memory Access area (the 'DMA') which is used to store records — either after a read operation or before a write operation. The default DMA location for CPN is 0080 hex. The DMA is relocated here on a cold or warm start, or a disc system reset. The DMA buffer consists of a single record of 128 bytes.

Function 27: Get Address (Allocation)

Passed Values	*Returned Values*
None	Registers HL: Alloc address

Function 27 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 could be inaccurate in the case of a write protected 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 not allocatable on the disc (i.e. the disc has a capacity of less than 4M bytes). The set bits 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

Function 28 gives temporary write protect status for the currently selected disc, until the next cold or warm start. Attempts to write to a disc with such protection will fail.

Function 29: Get Read/Only Vector

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

Function 29 returns a bit vector in registers HL to indicate drives which have the temporary read/only bit set. A set bit, '1' indicates read/only, an unset bit, '0' means read/write status.

As in function 24, the least significant bit of register L refers to drive A: while the most significant bit of register H refers to drive P:.

Function 30: Set File Attributes

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

Function 30 can be used to set the top bits of bytes f1–f8 and t1–t3 of an FCB.

This function is particularly useful for setting the read/only (t1 top bit) and system (t2 top bit) bits. 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 exchanged with the one addressed by the DE register.

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.

Function 31: Get Address (Disc Parameters)

Passed Values	*Returned Values*
None	DPB address

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

Function 31 returns the base address of the TCK (TORCH Control Kernel) resident DPB (Disc Parameter Block) in registers HL. The values in the block are available for the computation and display of useable space and for the recording of changes in the disc environment.

The function is provided for compatibility with the CP/M operating system.

Function 32: Set/Get User Code

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

Function 32 can be used to find the current user number or to change the user number. 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

Function 33 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 used to compute file size (function 35) or is otherwise zero.

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 BDOS is called to read the record into the buffer at the start of the DMA area. The record number, unlike sequential reading, is not updated with each operation. However, the ex and cr bytes are set with each read operation to the values for the record . It is therefore possible to read sequentially, commencing from a randomly accessed record. 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 unsuccessful, the following values are returned: 01 hex indicates that an unwritten record has been accessed; 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

Function 34 writes a record of data from the DMA address to the disc. As in function 33 (Read Random), bytes r0 to 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

Function 35 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. With a randomly written file, some sequential disc space is unallocated and the file may contain fewer records than indicated.

For example, if a file only contained record 65535 written in random mode, the size of the file would be given as 65536 records, even though it only contained one record.

The function searches for a match for the FCB addressed by registers DE. If byte r2 is set to 01 hex, then the file contained the maximum of 65536 records. Otherwise, bytes r0 and r1 contain the file size as a 16 bit value, with r0 as the least significant byte.

Function 36: Set Random Record

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

Function 36 returns the random record to which a file has been sequentially read/written. This has two main uses:

1. To produce a look-up table of the position of various keys in a sequentially read file, or
2. to change between random and sequential reading.

The file is identified by an FCB addressed by registers DE. Bytes r0 and r1 of this FCB are set to the random record position last read from or written to.

Function 37: Reset Drive

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

Function 37 resets specific disc drives, as indicated by the bit map in registers DE. The least significant bit indicates drive A:, while the most significant bit corresponds to drive P:. A set bit ('1') shows that a drive is to be reset.

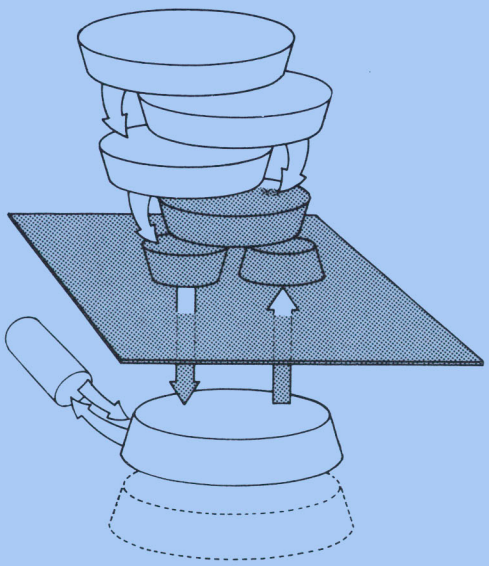
A value of zero is returned in register A unless the disc has open files on it that have been modified — in which case a non-zero result is returned.

Function 40: Write Random With Zero Fill

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

Function 40 is identical to function 34 (Write Random). It is provided for compatibility with systems which allocate records in groups rather than singly.

TORCH
COMPUTERS



Use of the Torch Base Processor

Use of the TORCH Base Processor

4.0 Contents

4.1 Introduction

4.1.0 The TORCH Base Processor

The TORCH Base Processor for current releases of the TORCH derives from the main board of the Acorn/BBC microcomputer (Model B) — the CPU of which is a 6502A. All the features normally available on the BBC microcomputer are accessible to the programmer, either through calls to the CPN Operating System or through the commands detailed in this section.

The facilities described in this section depend on hardware and firmware details of the current TORCH Base Processor (see the note on future upgrades below). These direct commands are outlined for the convenience of programmers only, as changes to the Base Processor hardware or its low-level software are outside the control of TORCH Computers Ltd. It is for this reason that all substantial applications programs written for the TORCH should, wherever possible, access the Base Processor only through the higher-level methods described earlier. TORCH Computers Ltd. will endeavour to maintain these interfaces wherever possible for all future combinations of Base and Applications Processors.

4.1.1 Debug Status — the '[System' Command

The TORCH provides a debug 'trace' facility available from the CCCP. This provides the same features as User Command 10 — Select Debug Status (see Section 4.4.2 below), but with the advantage that it may be set from the command line.

The command is of the form:

[system <number><return>

The <number> is given as a decimal value (base 10)and produces the trace effects as detailed in the following table:

Number	Set bit	Effect
0	none	Restore debug status (i.e. trace off)
2	1	Alter resets and ignore CTRL-SHIFT-ESC
4	2	Verify after a write operation
8	3	Display TORCHNET traffic
16	4	Display 'soft' errors, not normally reported
32	5	When a disc is being used in a read or write operation, the sector number being used is printed to the screen, with an 'r' for read or a 'w' for write. Other disc traffic (such as caching) is also displayed.
64	6	On a disc error, there is no hobnailed boot. This feature is reserved for test use.
128	7	All bytes passed between the Base Processor and the Applications Processor are traced on the screen.

4.1.2 Note on Future Upgrades

In addition to changes or developments of the Base Processor board outside of the control of TORCH Computers Limited, the company reserves the right to change the Base Processor hardware in future models of the TORCH Computer.

No undertaking is made to continue to support the direct interface for future versions of the TORCH. Wherever possible (which will be the great majority of cases), the higher-level access methods described in the earlier parts of this guide should be used to achieve the end effect required. In this way, the investment in software development effort should be protected, even if considerable changes are subsequently made in the TORCH design.

4.2 Technical Terms

Cutdown FCB

This is a File Control Block which is 12 bytes long rather than the 36 bytes of the full FCB. The first byte is the disc drive code, with 0 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 exactly 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

A file handle identifies a particular file by giving its position in an internal table. The file handle may have a value from 0 to 255.

Hard Boot

The Base Processor is reset by pressing the master reset button. All action is halted. Note that this can be dangerous if, for instance, the Base Processor is in the middle of writing a disc track. All Applications Processor memory will be uninitialised, save for CPN and CCCP, which are reloaded from ROM.

Cold Boot

CPN code and CCCP are reloaded into Applications Processor memory. All remaining Applications Processor memory is uninitialised. The Base Processor 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 Applications Processor 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	Invalid user function attempted.
0E	TORCHNET command attempted when a network is not connected.
55	Invalid file handle specified (usually due to using an FCB for transput without opening the corresponding file).
66	The Base Processor failed to find enough store to load a file structure block.
77	A disc transfer was incomplete (disc controller timeout).
A8	Invalid user function attempted.

Note: Results 66 and 77 are unlikely to occur unless the disc controller parameters are explicitly changed (see the TORCH Systems Manual for details of how to perform this).

A Cold Boot is carried out following the message display.

Warm Boot

CCCP is reloaded from ROM into Applications Processor RAM. All other contents of the application Processor RAM are preserved. The Base Processor is unaffected.

4.3 TORCH Base Command Interface

4.3.0 Accessing Interprocessor Commands

The TORCH Base commands provide a low level interface (below CPN and SUPERVDU) between the programmer and the TORCH Base Processor. A typical command will consist of the Applications Processor sending the number of the desired function to the Base Processor. There will usually be an exchange of information between the Applications Processor and the Base Processor. Note that in the first section of each specification where the values to be exchanged are given, there is no mention of the order in which they are to be passed. This order is given in each description below. Note that in some cases, not all the values which need to be given are passed.

For example, to Peek into the RAM controlled by the Base Processor at location 8000 hex, the Applications Processor would send:

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

The Base Processor would reply by sending the contents of address 8000 hex to the Applications Processor.

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

There are four 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 0FFC0 hex 

0F hex 

Sends 0F hex to the Base Processor followed by the byte given as an argument to the call. This is to allow easy calling of 'user functions' in assembly language. All Applications Processor registers apart from AF are preserved.

CALL 0FFC3 hex 

Sends the byte following the call to the Base Processor. Again, all Applications Processor registers apart from AF are preserved.

CALL 0FFC6 hex 

Returns with a byte in Applications Processor register A received from the Base Processor.

CALL 0FFC9 hex 

Sends the byte in Applications Processor register C to the Base Processor. Applications Processor registers AF are corrupted.

The last two routines above are represented throughout this manual by rx and tx, respectively.

4.3.1 TORCH Base Command List

Of the 26 TORCH Base Commands listed below, half of them (13) are enclosed in brackets. The 'hidden' Base Commands are only useful to the CPN Operating System and should not be used directly. Some information on these functions is given below, but this is intended to be of help in diagnosis, if required.

Function	Name
0	Hard reset Applications Processor
1	Print Byte
[2	Open File without extent]
[3	Close File without extent]
[4	Search For First without extent]
[5	Search For Next without extent]
[6	Delete Files]
[7	Read Record]
[8	Write Record]
[9	Create File without extent]
[A hex	Rename File]
[B hex	Set File Attributes]
[C hex	Get File Size]
D hex	Peek Into RAM
E hex	Poke Into RAM
F hex	Execute User Function
[10 hex	Set/Get User number]
11 hex	Get Keyboard Status
12 hex	Select Input Device
13 hex	Select Output Device
14 hex	Call Communications address
15 hex	Console Output (display byte)
16 hex	Get keyboard Status
[17 hex	Get Extent Size]
1C hex	Get Disc Configuration
1E hex	Execute TORCHNET Operation
Else	Invalid Call

4.3.2 Command Specifications

Command 0: Hard Reset Applications Processor

Passed Values
None

Returned Values
None

On receiving TORCH base command 0, the Base Processor issues a reset pulse to the Applications Processor, 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 Applications Processor.

The Base Processor then awaits further commands.

Command 1: Print Byte

Passed Values
ASCII character to be printed

Returned Values
None

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

[Command 2: Open File]

Passed Values
Cutdown FCB

Returned Values
Return code, file handle, file opened.

For Information only

The Base Processor receives a cutdown FCB (as described earlier). Using the disc drive qualifier, the appropriate disc directory is scanned for the first file match.

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

Should the file handle table be full, a code of 00 is also returned.

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

[Command 3: Close File without extent]

Passed Values
Cutdown FCB

Returned Values
Return code

For Information Only

A cutdown FCB is accepted from the Applications Processor 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
Cutdown FCB

Returned Values
Return code, file name, user number.

For Information Only

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

On finding a match, a return code of 00 hex is issued. The name of the matching file is then sent back to the Applications Processor, followed by the user number of the file matched.

If no file is found which matches the cutdown FCB, a return code of FF hex is returned.

[Command 5: Search For Next without extent]

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

For Information Only

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

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

[Command 6: Delete Files]

Passed Values	*Returned Values*
Cutdown FCB	Return code

For Information Only

This command accepts a cutdown FCB from the Applications Processor. All matching files are found and removed from the disc, along with all associated records and file structures.

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

[Command 7: Read Record]

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

For Information Only

The Applications Processor sends the Base Processor a file handle and reads the first open file found that matches. If the handle does not refer to an open file, the Base Processor issues a hobnailed boot. Otherwise, the Base Processor reads a word (a two byte pair) from the Applications Processor and reads the record at that position in the file, if possible.

If the record exists, the Base Processor returns a code of 00 hex, followed by the 128 bytes of the record. If the record has not yet been written, no record is passed and the return code 01 hex is issued.

[Command 8: Write Record]

Passed Values
File handle, record number, record

Returned Values
Return code

For Information Only

The Applications Processor sends the Base Processor a byte indicating which file handle to use. This is followed by a word (2 bytes) corresponding to a record number. If the Base Processor responds with 00 hex, the Applications Processor sends 128 bytes of data to make up the record. A response of FF hex indicates a failure to write the record.

The writing of a record may require several disc accesses if the file needs extending. Applications Processor processing resumes before the last disc access finishes.

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

[Command 9: Create File without extent]

Passed Values
Cutdown FCB

Returned Values
Return code, file handle.

For Information Only

The Applications Processor sends a cutdown FCB to the Base Processor, specifying the name of a file which it wishes to create.

A return code of FF hex is issued if:

1. A file with a matching name already exists on the disc.

2. The directory is full (it holds up to 256 names), or,

3. There is insufficient room on the disc for the file structure.

Otherwise the name of the file is entered in the directory of the appropriate disc and an empty file is assigned.

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

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 (A hex): Rename File]

Passed Values
Current cutdown FCB, new cutdown FCB.

Returned Values
Return code

For Information Only

The Base Processor accepts the current cutdown FCB from the Applications Processor followed by 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 hex): Set File Attributes]

Passed Values
New cutdown FCB

Returned Values
Return code

For Information Only

The Base Processor accepts a cutdown FCB from the Applications Processor. 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 the one 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 hex): Get File Size]

Passed Values
Cutdown FCB

Returned Values
Return code, size (2 bytes)

For Information Only

The Base Processor accepts a cutdown FCB from the Applications Processor and finds the first matching filename in the directory.

A code of FF hex is returned if no match was made. A return code of 01 is issued if the file is full (i.e. it has a length of 64K records).

If the operation was successful, a return code of 00 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 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 hex): Peek Into RAM

Passed Values	*Returned Values*
Address	(Contents)

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

Command 14 (E hex) Poke Into RAM

Passed Values	*Returned Values*
Address, data	None

The Applications Processor sends a 2 byte address (as low/high value pair of bytes), followed by a single byte, which the Base Processor writes or 'pokes' into that location.

Command 15 (F hex): Execute User Function

Passed Values	*Returned Values*
User Function number	<varies>

User Functions are special commands provided for TORCH systems programming work. The command is executed by the Applications Processor upon issuing the appropriate User Function number.

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

[Command 16 (10 hex): Set/Get User Number]

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

For Information Only

The Applications Processor sends a single byte to the Base Processor. If the value is FF hex, then the Base Processor returns a byte giving the currently set user number. Otherwise it sets the user number to the byte sent by the Applications Processor.

The initial user number after a cold boot is zero.

Command 17 (11 hex): Get Keyboard Status/Input

Passed Values	*Returned Values*
None	Return code, ASCII character

If no console input is pending, the Applications Processor returns 00 hex. FF hex, followed by the ASCII code is returned if console input is pending.

Codes other than 00 hex or FF hex for the first byte are reserved for future system development use.

Command 18 (12 hex): Select Input Device

Passed Values	*Returned Values*
Console input code	None

The byte sent by the Applications Processor following this command indicates which device should be treated as console input. The following selections are available:

0 Read the keyboard and ignore RS423 input. (The RS423 buffer is unaffected)

1 Lose the keyboard input. If the RS423 buffer is empty, read directly from the RS-423 serial line. Otherwise, the input from the RS 423 line is placed in an intermediate buffer storage and read indirectly.

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

All other selections result in undefined actions.

Command 19 (13 hex): Select Output Device

Passed Values	*Returned Values*
Console output code	None

The byte sent by the Applications Processor following this command indicates which device should be treated as console output. The following selections are available:

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

All other selections result in undefined actions.

Command 20 (14 hex): Call Communications Address

Passed Values	*Returned Values*

Communication Command Code

This command is used to interface to the asynchronus communications channel on the TORCH Base peripheral processor. The byte sent following the command controls the action taken.

The following communications commands are defined so far:

00 — Poll communications interface
01 — Perform interrupt action

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

For more information see the TORCH Communications Technical Manual.

Command 21 (15 hex): Console Output (display byte)

Passed Values	*Returned Values*
ASCII character	None

The Applications Processor sends a byte to the Base Processor which is sent to the console output channel.

Command 22 (16 hex): Get Keyboard Status

Passed Values	*Returned Values*
None	Return Code

The Base Processor 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 (17 hex): Get Extent Size]

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

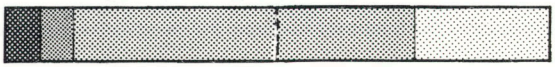
For Information Only

The Applications Processor sends the Base Processor a file handle to indicate which file is 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 Applications Processor sends the Base Processor an extent number. The Base Processor returns the number of records in that extent (from 0 to 127).

Command 28 (1C hex): Get Disc Configuration

Passed Value	*Returned Values*
Disc drive number (0 to 15)	0 = success; else = fail

If a value of '0' is returned, a 'disc configuration word' is also handed back. This word has the following structure:



15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

bits 00 to 03
: These contain the largest sector number on any track. Combined with bits 04 to 13 (see below), this forms the disc addressing information.

bits 04 to 13
: These give the largest track number on the disc. The largest track and sector (see above) numbers combined give the disc addressing information.

bit 14

Holds the 'local/remote' drive information. A value of '0' indicates a local drive. A value of '1' indicates that the drive is attached to a remote drive, via TORCHNET.

bit 15

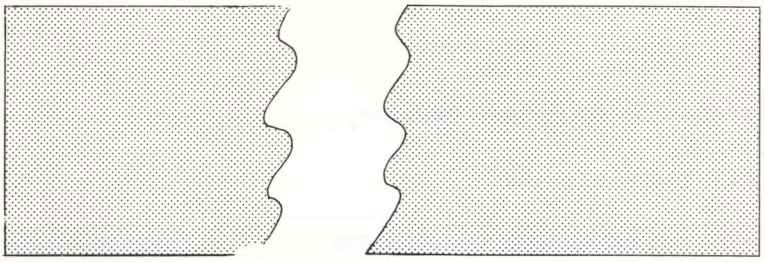
Indicates whether the disc medium is fixed or removeable.

The associations are:

0 = removeable drives
1 = fixed drives

Combining all the data in the disc configuration word, a rectangular matrix of addressable disc space may be constructed. For example, a removeable 5.25" flexible disc connected to a local drive (i.e. not connected to TORCHNET) might return a disc configuration word of 09F9. The least significant nibble ('9') gives a vertical axis from 0 to 9, the largest track number ('9F'), forms the horizontal axis.

So the word '09F9' corresponds to the following address matrix:



In the same way, a 10 MByte Winchester disc would return a disc configuration word of '9FFF' for a 2 MByte surface, or 'BFFF' for a 4 MByte surface.

If attached via TORCHNET, the values of the disc configuration word would be '49F9' for a floppy disc, 'DFFF' or 'FFFF' for a 2 MByte or 4 MByte surface hard disc respectively.

Command 30 (1E hex): Execute TORCHNET Operation

See the TORCHNET System Manual for information on this command.

4.4 TORCH Base User Functions

4.4.0 Accessing User Functions

All User Functions are accessed by calling TORCH Base command 15 (Call User Function) followed by the number of the required function to the Base Processor.

Thus: tx 0F hex, tx 00 hex

would call User Function 0 (Read Sector)

or: tx 0F hex, tx 08 hex

would call User Function 8 (Format Disc).

User Functions may also be accessed through the BIOS vector by calling location FFC0. The byte following the call contains the call number.

For example:

```
CALL    0FFC0 hex
DB      01 hex
```

would also call User Function 1 (Write Sector).

4.4.1 User Function List

Function	Name
0	Read Sector
1	Write Sector
2	Format Track
3	Select Disc
5	Get Escape Status
6	Copy 40 Bytes
7	Get Version Number
8	Format Disc
9	Unslave Disc Caches
A hex	Set Debug Status
B hex	Get Character Definition
C hex	System Call (Osword)
D hex	Read Scratchpad
E hex	Write Scratchpad
F hex	System Call (Osbyte)
10 hex	Toggle Printer Status
12 hex	Get Retry Count
15 hex	Call Base Processor Function
16 hex	Call Base Processor Subroutine
17 hex	Reset Handle Table
18 hex	Get Boot Control Byte
Else	Invalid Call

Note that User functions 4, 17 (11 hex) and 19 (13 hex) are not currently supported by TORCH Computers Ltd. The results which these calls were designed to produce are more reliably obtained through standard CPN calls.

4.4.2 User Function Specifications

User Function 0: Read Sector

Passed Values	*Returned Values*
Block number	(Block)

The Applications Processor sends the Base Processor a block number which 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.

For more information, see the Systems Manual.

User Function 1: Write Sector

Passed Values	*Returned Values*
Block number, block	None

The Applications Processor sends the 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 Applications Processor are written to the specified block.

For more information, see the Systems Manual.

User Function 2: Format Track

Passed Values	*Returned Values*
Track/side number	Return code

The Applications Processor sends the Base Processor a byte, whose least significant bit contains the side number and whose other seven bits contain the track number. If this value is illegal, a hobnailed boot is issued. Otherwise, the Base Processor attempts to format the specified track. A return code of 00 hex indicates success. Other possible 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 'lost' data was found on the disc.

User Function 3: Select Disc

Passed Values	*Returned Values*
Drive code	None

The Applications Processor sends the Base Processor a byte to indicate which drive to select. 00 hex represents drive A:, 01 drive B:, etc., up to 0F hex which represents drive P: on a 16 drive system.

User Function 5: Get Escape Status

Passed Values
None

Returned Values
Status

The Base Processor returns the Applications Processor a byte. If the top bit is set, then 'escape' has been pressed but not read (i.e. it is in the type-ahead buffer). Otherwise, the top bit is clear.

User Function 6: Copy 40 Bytes

Passed Values
Address

Returned Values
None

The Applications Processor sends the Base Processor a low/high pair of bytes giving an address in Base Processor memory. The next 40 bytes of data, starting at the next address (i.e. given address plus one), are moved 40 bytes towards high memory. If the address value causes 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 Function 7: Get Version Number

Passed Values
None

Returned Values
Version number

The Base Processor returns a byte representing the version of the TORCH Base ROM currently installed. Note that this is not necessarily the same as the current version of the operating system, CPN.

Thus the byte 60 hex would be returned for version 0.60.

User Function 8: Format Disc

Passed Values
None

Returned Values
Return code

(See also User Function 2: Format Track.) 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 'lost' data was found on the disc.

User Function 9: Unslave Disc Caches

Passed Values
None

Returned Values
None

All disc information in the memory of the TORCH Base Processor that is different from that on disc is written to the disc and is then erased from TORCH Base memory.

User Function 10 (A hex): Set Debug Status

Passed Values
Debug vector

Returned Values
None

The Applications Processor sends a single byte debug vector to the Base Processor, which controls both the current debug mode and the numeric keypad. Each bit has the following effect:

Number	Set bit	Effect
0	none	Restore debug status (i.e. trace off)
2	1	Alter resets and ignore CTRL-SHIFT-ESC
4	2	Verify after a write operation
8	3	Display TORCHNET traffic
16	4	Display 'soft' errors, not normally reported
32	5	When a disc is being used in a read or write operation, the sector number being used is printed to the screen, with an 'r' for read or a 'w' for write. Other disc traffic (such as caching) is also displayed.
64	6	On a disc error, there is no hobnailed boot. This feature is reserved for test use.
128	7	All bytes passed between the Base Processor and the Applications Processor are traced on the screen.

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

User Function 11 (B hex): Get Character Definition

Passed Values
ASCII character code

Returned Values
Character definition

The Applications Processor sends a byte to the Base Processor, giving the ASCII character code requested. The Base Processor returns a series of eight bytes, which indicate 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 for the same passed value, i.e. the character definitions from an absolute address in ROM firmware. To read the user defined characters, Osword Call 10 (Get Character Software Definition) should be used.

User Function 12 (C hex): System Call (Osword)

Passed Values
Osword call number

Returned Values
<varies>

This provides an interface to the Acorn MOS (Machine Operating System) Osword calls. The call is determined by the byte passed by the Applications Processor. For more information of Osword calls and ways of using them, see section 4.5 (Osword Call Interface).

User Function 13 (D hex): Read Scratchpad Byte

Passed Values	*Returned Values*
Byte number	(Byte)

The Applications Processor passes a byte number to the Base Processor, (from 0 to 3F hex) which is used as a displacement from the start of the TORCH Base Scratchpad (see section 4.5.1). The byte stored at this address is returned to the Applications Processor by the Base Processor.

User Function 14 (E hex): Write Scratchpad Byte

Passed Values	*Returned Values*
Byte number, byte	None

The Applications Processor passes a byte number to the Base Processor, (from 0 to 3F hex) which is used as a displacement from the start of the TORCH Base Scratchpad (see section 4.5.1). The byte stored at this address is returned to the Applications Processor by the Base Processor.

User Function 15 (F hex): System Call (Osbyte)

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

The Applications Processor passes the Base Processor an Osbyte call number followed by two parameters. The Base Processor performs the call and then returns a two byte result. (See section 4.6: Osbyte Call Interface for a list of Osbyte calls and their specifications.) The result may or may not contain useful information, depending on the call.

User Function 16 (10 hex): Toggle Printer Status

Passed Values	*Returned Values*
None	None

If the logical printing device is not being used for output, it is put on stream. If it was being used for console output, then it is switched off stream. For those readers familiar with CP/M, this is equivalent to sending the code '^P' (i.e. CTRL P) from the keyboard.

User Function 18 (12 hex): Get Disc Retry Count

Passed Values	*Returned Values*
None	Disc Retry Count

The Base Processor sends the Applications Processor a count of the unsuccessful attempts to access a disc. The count is initialised on power up to an undefined value and then incremented for every unsuccessful access. The value is passed as two bytes, with the least significant byte first. Note that if an error code of 01 hex is issued by the Base Processor 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.

This function is useful when called from within a user program to check the disc access reliability. Storing the value of the retry counts on power up and comparing the count after a number of disc accesses have been made will give the required data.

User Function 20 (14 hex): Get Fake Allocation Map

Passed Values	*Returned Values*
Disc number	Allocation vector

The Applications Processor sends the Base Processor a disc number with 00 hex corresponding to disc A:, through to 0F hex for drive P: (on a 16 drive system). The Base Processor returns a 32 byte value, representing the amount of space used on the disc. One bit shows 16k bytes used. The value returned will be either a stream of ones, indicating that space is used or was not accessible on the disc — it being smaller than 4M bytes — or a stream of zeroes, indicating unused space.

Note that no information about the actual disposition of data on the disc is given or implied.

User Function 21 (15 hex): Call Base Processor Function

Passed Values	*Returned Values*
4 byte address followed by 1 byte argument	1 byte result in reg A

This function allows the programmer's own code to be 'poked' into the Base Processor. The code may then be called at the address given and the value is returned in register A.

User Function 22 (16 hex): Call Base Processor Subroutine

Passed Values	*Returned Values*
4 byte address followed by 1 byte argument	None

This function has the same effect as User Function 21 (15 hex), except that no result is returned.

User Function 23 (17 hex): Reset Handle Table

Passed Value	*Returned Values*

This function should only be used when no disc files are currently in use, since all information in the Base Processor concerning the CPN filing system is destroyed. It will not close files before losing handle table information, so these should be closed explicitly by a CPN 'close' command.

This command may be useful because the maximum number of files which may be opened in a transient program is limited to 127.

A schematic example of how the function is used within an application program is as follows:

```
open 100 files
use these files
close all 100 files
reset handle table (i.e. User Function 17 hex)
open another 100 files
...
... and so on.
```

Note: The CCCP carries out a Reset Handle Table at the end of each command line. Therefore, 'copy' can handle an arbitrarily large number of files because a new command line is generated after each individual copy operation.

User Function 24 (18 hex): Get Boot Control Byte

Passed Values

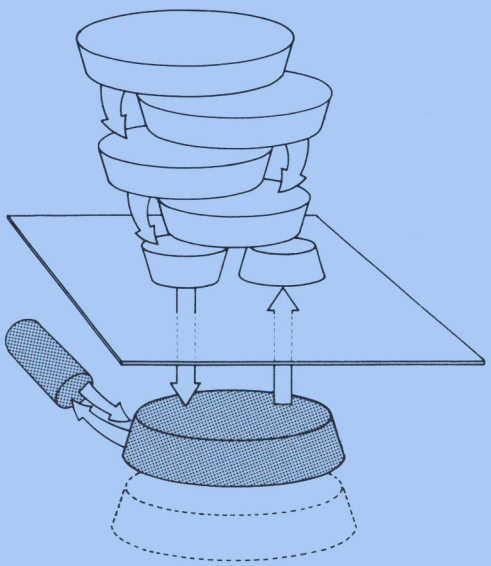
Returned Values
Boot Control Byte

The Boot Control Byte is used by the CCCP to determine certain actions following a 'cold' boot (i.e. after CTRL C or power up). The structure of the byte is of the form:

e	e	e	e	d	d	d	d

The top nibble (eeee) gives the TORCHNET status. A value of 0000 means that the Base Processor will not support TORCHNET calls. This is either because the Base Processor board does not support TORCHNET, or the TORCHNET hardware has failed. Any other value indicates a functioning TORCHNET, in which case, CPN and CCCP will automatically be replaced by the CPNet Operating System and CNCP Command Processor respectively.

The least significant nibble (dddd) represents the drive, from A: to P:, that will be selected by the console command processor after a cold boot. On a twin floppy disc system, this will normally be drive A:. On a hard disc system, a specific hard surface will be selected. (This will normally be drive B: for a single floppy disc/hard disc configuration, or drive C: in the case of a double floppy disc/hard disc set up.)



Acorn MOS Interface

Acorn Mos Interface

5.0 Contents

5.1 Acorn MOS Interface

The Acorn MOS (Machine Operating System) is a machine code program of about 16 Kbytes in size. It provides a 'soft' interface to handle many of the I/O devices — the keyboard, the interprocessor, graphics display, VDU drivers, analogue to digital convertor and sound generator. TORCH provides interfaces to some of the commands in the Machine Operating System.

The features of the Acorn MOS are described in detail in the BBC User's Guide. This section merely serves as a cross-reference to the commands and may be skipped by programmers not intending to use the interface.

Wrch	This is provided by TORCH Base Command 21 (15 hex), i.e. TX 15 hex, TX ch, where 'ch' is the hexadecimal code of the character to be printed; 'TX' is a call to the BIOS vector at 0FFC9. It was introduced in Section 4.3.0.
Osbyte	See 'Osbyte Call Interface' (section 5.2) for further details
Osword	See 'Osword Call Interface' (section 5.3) for further details.
Oscli	This command, to the MOS Command Line Interpreter, is invoked by Osword call 0. See 'Osword Call Interface' (section 5.3) for more details.
Scratchpad	See 'Osword Call Interface' (section 5.3.1) for further details of this interface.

5.2 Osbyte Call Interface

5.2.0 Accessing Osbyte Calls

Osbyte calls are accessed indirectly. First call TORCH Base Command 15 which will initiate User Command 15. The Osbyte call number is given next.

For example, to invoke Osbyte call nn, the following sequence of calls must be made:

TX 0F hex, TX 0F hex, TX nn hex.

Parameters for the Osbyte calls are passed in the next two bytes, in the order x, y.

A most important point is that Osbyte calls always issue two result bytes to the Applications Processor. On occasion the result may contain useful information; but in any case, undefined values are still returned. The Applications Processor should always reply with:

RX xx hex, RX yy hex

When the call is made, the system sets register A of the Base Processor to the call number, register X to the xx parameter and register Y to the yy parameter.

In addition, Osbyte calls may be accessed by '*FX' calls — either from the CCCP (the TORCH command line interpreter), from the Acorn CLI (if the computer is in BBC Basic mode), or by Osword call 0.

For example: *FX n,x,y

A slightly more complete list of *FX commands and Osbyte calls is given in the BBC computer's User Guide. The programmer interested in controlling the cassette motor, for instance, is directed to this publication.

When only one parameter ('x') has to be passed, the second parameter ('y') may be omitted. Note, however, that *FX commands are of no use where the Osbyte call returns a value, since the value is inaccessible to the *FX command.

5.2.1 Osbyte Call List

Decimal	Hex	Function
0	00	Return version number
2	02	Select input device
3	03	Select output device
4	04	Control cursor edit
5	05	Select printer
6	06	Set printer to ignore character
7	07	Set serial receive Baud rate
8	08	Set serial transmit Baud rate
9	09	Set flash mark period
10	0A	Set flash space period
11	0B	Set auto-repeat delay

12	0C	Set auto-repeat period
15	0F	Flush buffers
16	10	Select analogue to digital channels
17	11	Force analogue to digital conversion
18	12	Reset soft keys
19	13	Wait for field synchronisation
21	15	Flush selected buffer
117	75	VDU status byte
128	80	Read analogue to digital channel
129	81	Read keyboard with timeout
132	84	Get start of screen memory
133	85	Get start of screen memory for mode
134	86	Read cursor position
135	87	Read character at cursor position
137	89	Turn cassette motor On or Off
138	8A	Write to buffer
145	91	Read from buffer
156	9C	Set 6850 (ACIA) status register
188	BC	Get current Analogue to Digital conversion channel
189	BD	Get number of Analogue to Digital conversion channels
220	DC	Set escape character
225	E1	Set base number for function key codes
226	E2	Set base number for SHIFT function key codes
227	E3	Set base number for CTRL function key codes
228	E4	Set base number for SHIFT/CTRL function key codes
229	E5	ESC = 1B hex
231	E7	Enable or disable 6522 (VIA) IRQ
241	F1	Determine Osbyte call 1 (*FX 1) value
245	F5	Determine Osbyte call 5 value
246	F6	Determine Osbyte call 6 value

5.2.2 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 displayed on the screen which gives the version of the Acorn MOS (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.

The version number 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 Y = Undefined	Undefined

This call can be used by programs which require the use of the serial port for input.

The X parameter selects the device to be treated as console input. The following selections are available, all others causing undefined actions:

2,0 Read keyboard and enable the RS423 input. (The RS423 buffer is unaffected)

2,1 Lose keyboard input. If the RS423 buffer is empty, then read directly from the RS423 input stream; otherwise read from the buffer and buffer RS423 input.

2,2 Read from the keyboard and enable the RS423 input.

No useful value is returned.

Osbyte Call 3: Select Output

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

This call can be used by programs which require the use of the serial port for output.

The X parameter indicates where output is to appear on the three input streams. The following selections are available, all others causing undefined actions:

The console output code has the effect of selecting or deselecting the printer, screen or serial line respectively, as given in the table below:

	Printer	Screen	RS423
3,0	on	on	off
3,1	on	on	on
3,2	on	off	off
3,3	on	off	on
3,4	off	on	off
3,5	off	on	on
3,6	off	off	off
3,7	off	off	on

The 'Printer' is the selected printer device which of course can be configured for the serial or the parallel port. The default setting on power up is 3,0 (printer port and screen selected).

There is no useful value returned.

Osbyte Call 4: Control Cursor Edit

Passed Values
X = Control Code
Y = Undefined

Returned Values
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:

4,0 Resets the cursor editing keys to enable normal cursor editing. This is the default setting.

4,1 Disables cursor editing. The cursor control keys generate the following character codes:

Copy	135 (87 hex)
Left	136 (88 hex)
Right	137 (89 hex)
Down	138 (8A hex)
Up	139 (8B hex)

4,2 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

The keys may now be set by using *KEY 11,<string>, for example.

There is no useful value returned.

Osbyte Call 5: Select Printer

Passed Values
X = Printer code
Y = Undefined

Returned Values
Undefined

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

5,0	Printer Sink (i.e. not printed)
5,1	Parallel Centronics Port
5,2	RS423 Port
5,3	Selects a user-defined printer driver

The printer sink can be useful for avoiding a 'hung' system if no printer is connected, or the buffer is otherwise filled.

There is no useful value returned.

Osbyte Call 6: Set Printer to Ignore Character

Passed Values
X = Character
Y = Undefined

Returned Values
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 keyboard output to the printer (toggled by CTRL P). It has no effect on any CPN list functions.

This is feature can be very useful in suppressing an otherwise superfluous <linefeed> if the printer generates its own <linefeed> after every received <carriage return>.

This action may be carried out by Osbyte Call 6,10 (10 being the decimal code for ASCII <linefeed>).

Osbyte Call 7: Set Serial Receive Baud Rate

Passed Values
X = Rate code
Y = Undefined

Returned Values
Undefined

The Rx Baud rate for the RS423 Port is set to the value given by the X parameter. Valid codes are:

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

The highest Baud rate (19200) cannot be guaranteed and is not recommended. All other values cause an undefined action.

The standard transceiving format used by RS423 is:

1 start bit 8 data bits 1 stop bit

There is no useful value returned by the call.

Osbyte Call 8: Set Serial Transmit Baud Rate

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

The transmit Baud rate for the RS423 Port is set to the value given by the X parameter. The codes set up the same Baud rates as given for receive, Call 7, above.

No useful value is returned.

Osbyte Call 9: Set Flash Mark Period

Passed Values	*Returned Values*
X = Mark period	Undefined
Y = Undefined	

Physical Colour Codes 8 to 15 produce flashing colours. This call sets the time (in fiftieths of a second) spent displaying the first colour given in the list.

For example, Osbyte Call 9,5 will display the given colour for a duration of 0.1 seconds. The default is 9,25 (i.e. half a second).

No useful value is returned.

Osbyte Call 10: Set Flash Space Period

Passed Values	*Returned Values*
X = Space period	Undefined
Y = Undefined	

As above (Osbyte Call 9), this call sets the period, in fiftieths of a second, but for the complementary flashing colour.

There is no useful value returned.

Osbyte Call 11: Set Auto Repeat Delay

Passed Values	*Returned Values*
X = Delay period	Undefined
Y = Undefined	

The delay before a key auto repeats (i.e. repeatedly generates the keyboard code) while being depressed is set to the value (in centiseconds) given by the X parameter.

A value of zero (i.e. 11,0 completely 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	Undefined
Y = Undefined	

The period between each character being generated by auto repeats (measured in centiseconds) is controlled by the X parameter.

A value of zero (i.e. 12,0) 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	Undefined
Y = Undefined	

The buffers given by parameter X are flushed. The only valid codes are:

15,0	Flush all buffers
15,1	Flush the currently selected input buffer.

Any other value has an undefined effect. Values from 2 to 127 are reserved for future expansion. Values greater than 127 are 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).

To minimise ADC sampling delays (which depend on a minimum conversion time of 10ms per channel), programmers are advised to enable only those channels which are needed. Thus:

16,0 disables all ADC channels
16,1 enables channel 1
16,2 enables channel 1 and 2
16,3 enables channel 1, 2 and 3
16,4 enables all four ADC channels

There is no useful value returned.

Osbyte Call 17: Force Analogue to Digital Conversion

Passed Values
X = Channel number
Y = Undefined

Returned Values
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.

There is no useful value returned by this call.

Osbyte Call 18: Reset Soft Keys

Passed Values
Undefined

Returned Values
Undefined

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

There is no useful value returned.

Osbyte Call 21: Flush Buffer

Passed Values
X = Buffer code

Returned Values
Undefined

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

0 Keyboard buffer
1 RS423 input buffer
2 RS423 output buffer
3 Printer buffer
4 Sound channel 0 (noise)
5 Sound channel 1
6 Sound channel 2
7 Sound channel 3
8 Speech synthesis

There is no useful value returned.

Osbyte Call 117: Return VDU Status Byte

Passed Values	*Returned Values*
Undefined	X = VDU status byte

The VDU status byte returned contains the following information:

Bit

0	set if printer enabled (VDU 2), else 0
1	set if in page mode enabled (VDU 14)
3	set if software scrolling
5	set if cursors are joined (VDU 5)
7	set if VDU is disabled (VDU 21)

Osbyte Call 128: Read Analogue to Digital Channel

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

255	Keyboard
254	RS423 input
253	RS423 output
252	Printer
251	Sound channel 0
250	Sound channel 1
249	Sound channel 2
248	Sound channel 3
247	Speech

Osbyte Call 129: Read Keyboard with Timeout

Passed Values
X,Y = Maximum time delay

Returned Values
X = Character code
Y = Character detected flag

This is the call used by the BASIC function 'INKEY'. Programmers should remember to flush the keyboard buffer (using Osbyte Call 15,1) if the 'typed ahead' characters are not required.

The maximum time delay in centiseconds is given as a hex pair in the X,Y registers. The highest value is 7FFF hex (which allows a delay of over 5 minutes).

The Y flag is cleared to 0 if a character was detected within the timeout period (in which case the character is given to X). A value of 1B hex in Y indicates that <esc> was pressed; the value FF hex implies a timeout.

Osbyte Call 132: Get Start of Screen Memory

Passed Values
Undefined

Returned Values
X,Y = 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
X = Mode
Y = Undefined

Returned Values
X,Y = Start of screen memory

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 result of this call is undefined.

Osbyte Call 134: Read Text Cursor Position

Passed Values
Undefined

Returned Values
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 3.8 on Character Output). The top left corner is specified as 0,0.

Note that two dummy variable have to be transmitted to the Base Processor after the call number (134).

Osbyte Call 135: Read Character at Cursor Position

Passed Values
Undefined

Returned Values
X = Character code
Y = Current graphics display mode

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

Osbyte Call 138: Write To Buffer

Passed Values
X = Buffer code
Y = Character code

Returned Values
Undefined

This call can be used to write bytes to the serial output port.

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

0	Keyboard
1	RS423 input
2	RS423 output
3	Printer
4	Sound channel 0
5	Sound channel 1
6	Sound channel 2
7	Sound channel 3
8	Speech

Note that there is no check by this call to determine if the buffer is full. Osbyte Call 128 can be used for this purpose.

For example, to insert the character 'A' (which is ASCII code 'decimal 65') into the printer buffer, use Osbyte Call 138,3,65.

There is no useful value returned.

Osbyte Call 145: Read From Buffer

Passed Values
X = Buffer code
Y = Undefined

Returned Values
X = Undefined
Y = Character code

This call can be used to read bytes from the serial port.

The next character is read from the specified buffer and returned as the Y parameter. Valid buffer codes are listed in the description of Osbyte call 138, above.

Note that there is no way of determining when a buffer is empty with this function. The call should therefore be used in conjunction with Osbyte Call 128 (Read ADC channel).

Osbyte Call 156: Set 6850 (ACIA) Status Register

Passed Values
X = byte mask
Y = byte mask

Returned Values
X = Result
Y = Result

This call behaves like an an Osvariable (see, for example, Osbyte Call 221), but controls the status register of the 6850 (ACIA device).

Osbyte Call 188: Get Current Analogue to Digital Channel

Passed Values
Undefined

Returned Values
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
Undefined

Returned Values
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 0 to 4.

Osbyte Call 220: Set Escape Character

Passed Values
X = Character code
Y = Undefined

Returned Values
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 <esc> key will still retain 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
See below

Returned Values
X = Old value of Osvariable
Y = Undefined

There are 8 Osvariables in the system which are used to determine the handling of input codes 80 to FF hex from the keyboard or RS423. Each Osvariable is controlled by one Osbyte call and refers to a block of 16 codes, as follows:

221: C0–CF
222: D0–DF
223: E0–EF
224: F0–FF
225: 80–8F
226: 90–9F
227: A0–AF
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 interpreted by the MOS
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

The old value of the Osvariable is returned as the X parameter. No useful value is 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.

For example, the <begin>, <end>, etc. keys of the editing keypad normally produce codes in the range 90 to 9F hex. This is because Osvariable 227 has the default value of 90 hex.

Osbyte Calls 241 to 246: Osvariables

These Osvariables are set by Osbyte Calls 1 to 6. The correspondences are:

241		
242	Select input stream	(Osbyte Call 2)
243	Select output stream	(Osbyte Call 3)
244	Cursor key codes	(Osbyte Call 4)
245	Printer selected	(Osbyte Call 5)
246	Ignored print character	(Osbyte Call 6)

Thus Osvariable 245 may be used to reveal which type of printer is in use.

5.3 Osword Call Interface

5.3.0 Accessing Osword Calls

Osword calls are accessed indirectly. First use TORCH Base Command 15 which will initiate User Command 12. The call number which follows these two commands is passed by the Applications Processor to the Base Processor.

For example, to bring about Osword call nn, the following sequence must be made:

TX 0F hex, TX 0C hex, TX nn hex.

Parameters for the Osword calls are passed and returned in the TORCH Base Scratchpad.

When the call is made, the system sets the XY register pair of the Base Processor to point at the base of the scratchpad and the Osword call number is placed in register A. For more details, see section 5.3.1 below.

5.3.1 The TORCH Base Scratchpad

The Base Processor has a 40 hex (64 byte) area of memory, known as the scratchpad, which is used for passing parameters to and from Osword calls and for programs running on the applications processor.

Bytes are read from the scratchpad with user call 13; they are written to the scratchpad by user call 14. Thus, to read ('peek') the value mm hex of the nnth byte of the scratchpad, the following calls must be made:

TX 0F hex, TX 0D hex, TX nn hex, RX mm hex.

To write the value (i.e. to 'poke') mm hex to the nnth byte of the scratchpad, the following call sequence must be made:

TX 0F hex, TX 0E hex, TX nn hex, TX mm hex.

5.3.2 Osword Call List

Decimal	Hex	Function
0	00	**Pass Scratchpad to CLI**
1	01	**Read Absolute Time**
2	02	**Write Absolute Time**
3	03	**Read Interval Time**
4	04	**Write Interval Time**
7	07	**Make Sound**
8	08	**Define Envelope**
9	09	**Read Pixel**
10	0A	**Get Character Software Definition**
11	0B	**Read Colour Relationship**
13	0D	**Read Graphics Cursor Position**

5.3.3 Osword Call Specifications

Osword Call 0: Pass Scratchpad to CLI

Passed Values
Line to be interpreted

Returned Values

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 BBC Basic
*KEY	Redefines soft keys
*FX	Calls Osbyte. This is better done direct: see the above section, 5.2.

Osword Call 1: Read Absolute Time

Passed Values
None

Returned Values
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 Base Processor, 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.

'0' represents 1 January 1980 at 00:00:00.

Osword Call 2: Write Absolute Time

Passed Values
Absolute Time

Returned Values
None

This call writes the value of bytes 00 to 04 of the scratchpad to the absolute timer. Byte 00 is of course the least significant.

Osword Call 3: Read Interval Time

Passed Values
None

Returned Values
Interval Time

This call reads the value of the interval timer into bytes 00 to 04 of the scratchpad, with the least significant byte in byte 00. The interval timer is an internal clock on the Base Processor, 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
Absolute Time

Returned Values
None

This call writes the value of bytes 00–04 of the scratchpad to the interval timer, byte 00 being the least significant one.

Osword Call 7: Make Sound

Passed Values	*Returned Values*
8 Sound Parameters	None

Apart from defining an envelope for sound output, this call has an important use in accessing the speech synthesis unit.

A sound is defined from eight parameter bytes passed to the Base Processor from the scratchpad. The parameters are in the form of four byte pairs, (i.e. 'words'). Starting from byte 00 of the scratchpad, these are:

Channel word
Amplitude word
Pitch word
Duration word

In more detail:

Channel word (bytes 0 and 1)

byte 0:

The four least significant digits (i.e. bits 0 to 3) specify which of three sound channels or one noise channel are defined. A value of 0 defines the noise/pulse wave channel; values between 1 and 3 define the sound generating channels.

The most significant digits of byte 0 (i.e. bits 4 to 7) determine whether the sound is 'queued' or not. Up to four sound definitions may be queued in the sound buffer for each channel in addition to the sound being played. A value of zero will queue the definition; a value of 1 will cause the sound buffer to be flushed and the sound to be immediately output.

byte 1:

The four least significant bits are used to specify chord synchronisation, by indicating how many other sound channels have the same value. This digit must be ready before the sound is played and can have a value between 0 (the default value) and 3.

The four top bits of byte 1 are used to transmit a dummy sound to the channel, to continue the previous sound. A value of '1' indicates that the note is a dummy; a zero indicates that the note is to be played.

Amplitude word (bytes 2 and 3)

byte 2:

The amplitude (loudness) is specified in 2's complement form, using only the least significant five bits of this byte.

For sounds which require varying pitch and amplitude, use numbers between 0 and 15 to identify an envelope (see Call 8 'Define Envelope', below). Numbers from –16 to –1 will define a note of both constant pitch and amplitude.

byte 3:

Is not used.

Pitch word (bytes 4 and 5)

byte 4:

The interpretation of the pitch word depends on the setting of the least significant four bits of the 'channel word' (i.e. of byte 0, see above). If channel 0 has been requested, then noise is generated. The type of noise is controlled in turn by the setting of the least significant two bits of byte 4. Thus:

0	pulse wave
1	pulse wave
2	grey noise
3	frequency is linked to channel one (see channel word, above)

Otherwise, the value between 0 and 255 contained in byte 4 is simply taken to represent the desired pitch. This is scaled so that '1' represents the musical note B which is 13 semitones below middle C.

Pitch increases at the rate of one quarter semitone per digit; so four digits increment will represent a semitone rise, eight digits a full tone. In this way, decimal '53' will represent middle C.

byte 5:

Is not used.

Duration word (bytes 6 and 7)

byte 6:

Duration is given by the value passed over from byte 6. A decimal value between 0 and 254 will give the duration value in units of 50 ms (twentieths of a second). A value of '255' with all the bits set, however, will give a note without end.

byte 7:

Is not used.

Osword Call 8: Define Envelope

Passed Values	*Returned Values*
14 Envelope Parameters	None

The Define Envelope call controls the changes in the pitch (frequency) and the volume (amplitude) of a sound whilst it is playing. It is used in conjunction with Osword Call 7, 'Make Sound' (see above).

Changes in pitch (high notes and low notes) with time are controlled by a pitch envelope. Variations in amplitude are governed by an amplitude envelope, which imply three elements: an increase to the loudest point ('attack'), the length for which this loudest value is held ('sustain') and the change in volume ('decay').

The 'envelope' defines both pitch and amplitude changes with the aid of fourteen parameters. The fourteen bytes passed to the Base Processor from the scratchpad are:

Parameter	Range	
envnum	0 to 15	
step	0 to 255	
dpitch0	−128 to 127	'pitch envelope'
dpitch1	−128 to 127	
dpitch2	−128 to 127	
stepnum0	0 to 255	
stepnum1	0 to 255	
stepnum2	0 to 255	
attack−127 to 127		'amplitude envelope'
decay	−127 to 127	
sustain	−127 to 0	
release	−127 to 0	
attacklevel	0 to 126	
decaylevel	0 to 126	

In more detail the parameters are:

envnum:

This specifies the envelope number that is about to be defined.

step:

The step parameter determines the length of each step of both the pitch and the amplitude envelopes in centi-seconds (units of 0.01 seconds). Values between 0 and 127 will cause auto-repeat for the pitch envelope. If values greater than this are used (i.e. if the most significant bit of the 'step byte' is set), then the auto-repeat is suppressed.

dpitch0 to dpitch2 and stepnum0 to stepnum2:

Are the six bytes which define the three sections of the 'pitch envelope'. For each of these three sections there is a dpitch/stepnum pair which control the change in pitch and the number of steps in each section respectively. The change in pitch may be positive or negative. The pitch envelope starts from a datum line given by the 'pitch word' in Osword Call 7 ('Make Sound').

attack, decay, release, sustain:

The profile of the amplitude envelope is made up from these four parameters, which give the values of the attack, decay, release and sustain slopes. The slopes represent changes of amplitude per step for each of the four phases. If the decay rate is given as 0, then loudness will be maintained at the level set by 'attacklevel' (see immediately below).

attacklevel, decaylevel:

'Attacklevel' and 'decaylevel' are two highly significant points on the amplitude envelope. The amplitude envelope always starts at zero and rises to a target level (the 'attacklevel') — at a rate determined by 'attack', the attack rate (see above). 'Attacklevel' may vary between 126 (loudest) to 0 (silence). Similarly, the target level for the end of the decay phase is set by 'decaylevel' and falls (or, indeed rises, depending on the value relative to 'attacklevel') at a rate determined by 'decay' (see above).

Osword Call 9: Read Pixel

Passed Values
X, Y coordinates

Returned Values
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
ASCII character code

Returned Values
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, continuing to byte 8 which represents the bottom row from left to right.

A set bit ('1') in each byte indicates that the corresponding pixel is in the foreground colour. A clear bit ('0') puts the pixel in the background colour.

Osword Call 11: Read Colour Relationship

Passed Values
Logical colour

Returned Values
Corresponding physical colour

A logical colour code (modulo the number of colours in the current screen mode) is passed to byte 0 of the scratchpad. The corresponding physical colour code is returned in byte 1. The physical colour code relationships are:

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

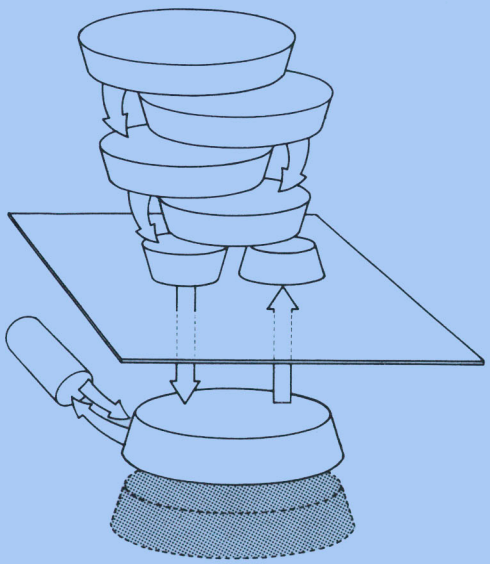
Bytes 2, 3 and 4 are zeroed. They are reserved for future expansion.

Osword Call 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 handed back in bytes 0 to 3 and its current position in bytes 4 to 7.

The screen has co-ordinates of 1280 x 1024, but the cursor may only be addressed at the coarser resolution of 640 x 256. The x value is divided by 2 and the y value by 4, with all values rounded down.



Direct Console Output

Direct Console Output

6.0 Contents

6.1 Console Output

This section details the effect of passing control characters to the screen. This may be done either:

i) via CPN
ii) via the BIOS vector,
iii) using the TORCH Base Command Interface, or
iv) from the CCCP by the VDU command.

When control codes cannot be satisfactorily issued from within applications programs, the SUPERVDU program should be loaded and used. See Section 7, 'SUPERVDU Functions'.

In more detail, the access levels are:

i) 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.

ii) From the BIOS vector, a character is sent by placing it in register C of the Applications Processor and making a call to location FFD8 hex. For example:

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

would send the <escape> character (ASCII 1B hex) to the screen. This method is not recommended, except in speed critical applications, since no error recovery or interpretation of the codes transmitted is carried out.

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

iv) From the CCCP, characters may be output directly to the screen by the VDU command. See the 'TORCH Systems Guide' for futher information. The values following the command are sent directly to the screen, for example:

VDU 3,4,0,10,27

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

Note that different control characters will call for varying numbers of parameters. All parameters called for must be sent, even if they do not seem relevant or have a null value.

6.1.1 VDU Output Code List

The table listed below is important. The five columns are organised in the following way: the first column gives the decimal ASCII character code involved; 'hex' gives the hex value of the ASCII character code, 'CTRL' gives the character which, when pressed with control held down, will produce the code given in the row. 'Bytes' give the number of parameters (in bytes) needed and 'function' gives the function name.

Brief specifications of each command follow this list.

Number	Hex	CTRL	Bytes	Function
0	00	@	0	<null>
1	01	A	1	Send next character to printer only
2	02	B	0	Enable printer
3	03	C	0	Disable printer
4	04	D	0	Separate text/graphics cursors
5	05	E	0	Join text/graphics cursors
6	06	F	0	Enable VDU Driver
7	07	G	0	Ring 'bell'
8	08	H	0	Cursor left one character
9	09	I	0	Cursor right one character
10	0A	J	0	Cursor down one line
11	0B	K	0	Cursor up one line
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 (gcol)
19	13	S	5	Define colour relationship
20	14	T	0	Reset colour relationships
21	15	U	0	Disable VDU Drivers
22	16	V	1	Select mode
23	17	W	9	Define character
24	18	X	8	Define graphics window
25	19	Y	5	Plot (m,x,y)
26	1A	Z	0	Reset windows
27	1B	[	0	<escape>
28	1C	/	4	Define text window (lhx,by,rhx,ty)
29	1D	]	4	Define graphics origin
30	1E	^	0	Home text cursor
31	1F	_	2	Position text cursor (x,y)
127	7F		0	Delete character

6.1.2 VDU Output Code Specifications

Output Code 0: <null>

This code has no effect on the screen display.

Output Code 1: Send Next Character to Printer Only

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 or 2 is sent to the VDU driver (see above).

Output Code 4: Separate Text/Graphics 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 Text/Graphics 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 One Character

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 One Character

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 One Line

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 One Line

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 toggles 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 Screen 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.

The 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	text only	80 x 25	2
4	320 x 256	40 x 32	2
5	160 x 256	20 x 32	4
6	text only	40 x 25	2
7	Teletext	40 x 25	16

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.

Any values not listed above are reserved for future expansion.

All 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). So in 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>

If the high-level SUPERVDU.COM program is loaded in the PLA, the various SUPERVDU functions may be invoked following the output from a program of the ASCII <escape> character (1B hex); see section 7, 'SUPERVDU Functions'.

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.

6.2 External Interfaces

Analogue to Digital Interface

The full specification of this interface will be described in the TORCH Communications Manual. The descriptions of Osbyte calls 16, 17, 128, 188 and 189 give relevant information.

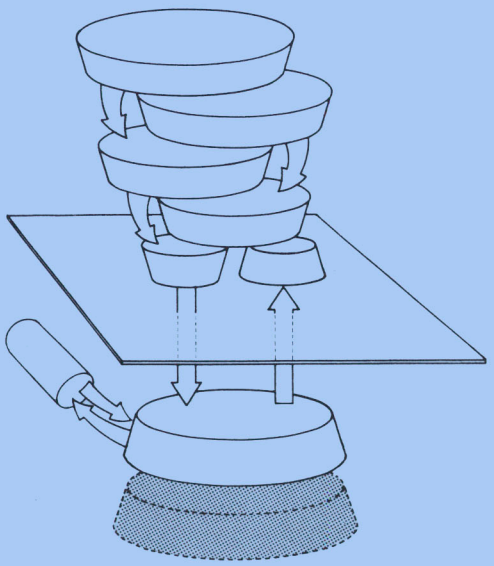
TORCHNET Interface

See the TORCHNET manual for details of this Local Area Network system.

Light Pen Interface

Details of the light pen interface are not available at the time of printing this edition of the Programmers' Guide.

TORCH
COMPUTERS



SUPERVDU Functions

SUPERVDU Functions

Contents

7.1 Introduction

7.1.0 Facilities

This is a guide to SUPERVDU controls. They are called from an applications program and are used for achieving both graphic and text output to the screen.

There are several available 'streams' of Input/Output allowing a choice of configurations. For the TORCH, the streams normally used are:

SUPERVDU	for text output to screen
Graphics	for graphics output to screen
Printer	for printed output

Facilities available include the following:

SUPERVDU Stream

1. Selection of different I/O streams (e.g. printer, VDU)
2. Panning the screen over a larger area 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. Panning the screen over a larger area 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.
2. Selection of Baud rate.

7.1.1 Accessing SUPERVDU Functions

7.1.1.1 Loading SUPERVDU

To use any of the facilities outlined in this section, the SUPERVDU program must first be loaded from the systems disc provided. The executable file is called 'SUPERVDU.COM' and is loaded in the same way as any .COM file; that is, you type:

```
<drive code>:supervdu
```

After the program has been loaded into memory, the disc may be removed.

An alternative to using the supplied program is to enter a filename as a parameter to SUPERVDU. The file will then automatically load and execute.

7.1.1.2 Invoking SUPERVDU Functions

Once the program is loaded, SUPERVDU functions are invoked by sending ESC ('escape', or ASCII code 27) from within a program followed by a single character to indicate the function. In addition, a set of arguments may be necessary. For instance, to move the cursor left from an MBASIC program, whilst in SUPERVDU stream:

```
LET ESC$ = CHR$(27)
LET CURSORLEFT$ = ESC$ + "W"
```

or in Z80 assembler, under CPN:

```
ESCAPE      EQU     1BH

CURLEFT:    LD      DE, CSRLEFT
            RST     0030H
            DB      9                   ;String output
            RET
CSRLEFT:    DB      ESCAPE,'W','$'
```

7.1.1.3 Argument Formats

Arguments passed to these functions are always numbers in decimal ASCII. In all cases (except relative cursor addressing and the plot commands in the graphics stream) any sign given will be ignored.

Each argument after the first number must be preceded by a separator, this being a sequence of one or more non-numeric characters. The last argument must be followed by a terminating non-numeric character. The entire escape sequence from <escape> to the terminating character will be interpreted unless an invalid combination is given, in which case only the following character will be acted upon. Note that a carriage return followed by a line feed are treated as a single character.

All parameters must be given. However, if any parameters specified are outside the range permitted then the entire function will be ignored.

For example, to clear a page and select the mode from MBASIC:

```
LET CPSM$ = ESC$ + "&"

PRINT CPSM$;1;1      ' Clear and select screen mode 1.
```

Note that the second '1' is terminated by the newline sent by MBASIC after each print statement and that the numbers are separated by the space output by MBASIC between numbers. Also note that an illegal screen mode or a page number other than one would cause the function to have no effect although the parameters would still be taken up.

7.1.1.4 Coordinates

Coordinates used for the SUPERVDU stream are measured in character positions and are therefore dependent on the mode set. A screen can be 80, 40 or 20 units wide and 32 or 25 units high. The origin of the page in every case has the coordinates (1,1).

7.2 Description of the SUPERVDU Stream

7.2.0 Implementation Restrictions

The SUPERVDU program is intended to allow the user to process text within large 'pages' of memory. The screen may be positioned over these windows allowing the user to view specific areas of text.This concept is restricted to a single page equal in size to the screen on the current implementation of SUPERVDU. It should therefore be remembered that although functions below referring to 'pages' are implemented, they usually have no effect (the exception is 'clear page'). Whenever a function requires a page number, the figure '1' should be given.

The currently used Base Processor unit does not support character insert and delete functions. Therefore, these functions (along with column insert/delete) have not been implemented.

Certain features do not operate within certain modes. For example, there are several restrictions on non-scrolling windows. In addition, non-graphics screen modes are unable to display 'in-character' underlining. For further details, see below.

7.2.1 Use of Windows

There is always a 'window' selected as the current window, and the cursor may not be moved out of it by using SUPERVDU functions. There is no need for the screen to display the current window or the cursor. The screen's movement is independent of the cursor's movement. All editing is restricted by the parameters of the current window. Whenever a new window is selected, the cursor is moved to its last position or the top left hand corner of the window if it has not yet been used. If the window is non-scrolling, then the cursor position will not be preserved when another window is selected.

Windows may be designated as scrolling or non-scrolling.

If an attempt is made to move the cursor off the top or bottom of a non-scrolling window, the cursor wraps round from the top to the bottom of the column it is in. If the cursor is moved off the end of a line, it is sent to the beginning of the next line down. Similarly, if the cursor is moved before the beginning of a line, it is sent to the end of the previous line.

When the cursor reaches either the start or end of a scrolling window, all text scrolls by one line up or down.

7.2.2 SUPERVDU Stream: Function List

Available SUPERVDU functions for the SUPERVDU stream are given in a list below and then described in detail in the next section. The individual functions are accessed by sending the appropriate character after an escape code.

General

<escape> (i.e. 1B hex) Select Stream
<delete> (i.e. 7F hex) Select debug mode
'f' Send star command to TORCH base.

Screen Selection

<space> Initialise
'!' 'Page Mode' On/Off
'$' 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
'N'	Delete Line
'O'	Insert Line

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

User defined Characters

'd'	Define Character
'e'	Direct character output

7.2.3 SUPERVDU Stream: Function Specifications

Select Stream

Called by: <escape><escape>

Parameters:

'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

This function is used to control the currently active streams to which SUPERVDU will send output. More than one device may be selected at a time. Note that this function is unusual in that it takes single character parameters. These single characters should immediately follow the last escape. The usual method of selecting a stream is to send 'A' as the parameter. If two streams are selected they will both receive all printing characters but escape sequences affect only one.

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	

Debug Toggle

Arguments:

<escape> <delete>, and

1	On, or
0	Off

When debug mode is selected all control characters are displayed in the form caret ('^') followed by the appropriate letter. For example, carriage return will be displayed as ^M. Note that 'delete' is 7F hex.

Send Star Command

Arguments:

<escape> 'f' command

The command is a string of not more than 63 characters terminated by a carriage return (and optional line feed). This command is sent to the TORCH Base processor and interpreted by the MOS command line interpreter. Examples of useful commands are:

*KEY and
*FX

The '*' character must be included among those sent. If the string is longer than 63 characters it will be truncated.

Initialise

Arguments:

<escape> <space>

This function sets 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.

'Page Mode' Toggle

Arguments:

<escape> '!''1'	On
<escape> '!''0'	Off

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

Select Page

Arguments:

<escape> '$' page number

This function currently has no effect.

Clear Page

Arguments:

<escape> '%' page Page number must be 0 or 1

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

Clear Page and Select Mode

Arguments:

<escape> '&' page, mode Page = 0 or 1, mode = 0 to 7

This function clears the page as above as well as selecting 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	text only	80 x 25	2
4	320 x 256	40 x 32	2
5	160 x 256	20 x 32	4
6	text only	40 x 25	2
7	Teletext	40 x 25	16

The new mode will have the default colours displayed (see 'Select Foreground Colour'). To change the colour mapping — that is, the relationship between the physical and logical colours — refer to the section: 'Define Colour Relationship'.

Position Screen Origin

Arguments:

<escape> '" x, y

Since the page size is equal to the screen size this function currently has no effect. This comment applies equally to the pan screen functions (i.e., horizontal scrolling) below.

Define Window in Page

Argument:

<escape> '0'

window number,
page number, x, y,
width, height,
scroll type Must be unique (see below)

A window is created at origin x, y on the given page with the specified height and width. The window is assigned a number. If this number has been used previously, then the window is redefined, given that the new arguments are legal. A 'scroll type' of one indicates a scrolling window, while zero indicates a non-scrolling window. All other scroll types will cause the window to become undefined.

Window 0 is used in the functions below to refer to the whole of the current page and may not be redefined. If a window number is selected that is greater than the maximum permissible one, then the function has no effect. There are currently 10 windows numbered 0–9 of which the last 9 may be redefined by the user.

A page number of zero indicates the current page. If a number is given which is greater than the number of pages, then the function will have 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. The function will have no effect if any of the arguments specify a window which is off the page.

Select Window

Argument:

<escape> '1' window number

The selected window is specified as the current window. All subsequent output will take place within this window until another command affecting window selection is given. If a scrolling type window has been selected the cursor will remain at its last position within the window. Otherwise the cursor is homed to the start of the window.

Window 0 always corresponds to the whole of the current page and always scrolls. If a window is selected that has not been defined, then the function has no effect.

Clear Window

Argument:

<escape> '2' window number

The specified window is cleared (i.e. it is made blank). If it is of scrolling type then the cursor position will be unaffected. The parameters of the currently selected window are unchanged by this command.

Clear to End of Window

Argument:

<escape> '3'

The window is cleared from the current cursor position to the end of the last line. The cursor is not moved. This function will not work in a non scrolling window.

Clear to Start of Window

Argument:

<escape> '4'

The window is cleared from the character immediately to the left of the cursor to the beginning of the first line. The cursor position is not moved. This function will not work in a non-scrolling window.

Cursor Address Relative in Window

Argument:

<escape> '<' x, y

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

Home Cursor in Window

Argument:

<escape> '='

The cursor is homed in the current window.

Cursor Address Absolute in Window

Argument:

<escape> '>' x, y

The cursor is moved to the absolute location within the window. Coordinates of 1,1 represent the origin of the current window.

Clear Line

Argument:

<escape> 'D' 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. The function will have no effect if a line number is given that is outside the current window.

Clear to End of Line

Argument:

<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 currently defined window inclusive.

Clear to Start of Line

Argument:

<escape> 'F'

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

Clear Characters Right

Arguments:

<escape> 'H' 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 cursor 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:

<escape> 'I' 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 cursor, 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 Line

Arguments:

<escape> 'N' line number

The given line is deleted and all lines below it in 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:

<escape> 'O' 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 to view. 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 is ignored.

Cursor Up

Arguments:

<escape> 'T'

The cursor is moved up the current window by one line.

Cursor Right

Arguments:

<escape> 'U'

The cursor moves one space to the right. It will wrap around to the start of the next line when it reaches the edge of the window.

Cursor Down

Arguments:

<escape> 'V'

The cursor is moved down the current window one line. If it reaches the bottom of a scrolling window the window will scroll. Otherwise the cursor will wrap around to the top of the screen.

Cursor Left

Arguments:

<escape> 'W'

The cursor moves one location to the left wrapping round and/or scrolling as necessary.

Select Foreground Colour

Arguments:

<escape> '\' 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:

<escape> ']' 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:

<escape> '^' logical colour code, physical colour code

The logical colour is the code actually stored for each pixel on the screen. The physical colour is the colour which appears on the screen.

The given logical colour code, modulo the number of colours in the current mode (i.e. 2, 4 or 16), is redefined according to the physical colour code given (see 'Select Foreground Colour'). All colour relationships apply only to the current mode and are cleared whenever the mode is changed.

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

Set Enhancement

Arguments:

<escape> '£' (i.e. 60H) enhancement mode

The list of enhancement modes is set to the enhancement passed as an argument. This new list is used until changed.

Currently available modes of enhancement are:

0	No enhancement
2	Inverse
5	Underlined

Add Enhancement

Arguments:

<escape> 'a' enhancement mode

The specified enhancement mode is added to the list of enhancement modes.

Remove Enhancement

Arguments:

<escape> 'b' enhancement mode

The enhancement mode passed to the function is removed.

Character Definition

Argument:

<escape> 'd'

character code, Range 32 to 255
character width, ┐ In — Must be 8
character height, ┘ pixels — Must be 8
row representation(s)

The ASCII code of the character which the user wishes to redefine is given, followed by its width and height and the set of row representations of the pixels making up the character.

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 unsigned decimal number and returned.

Direct Character Output

Arguments:

<escape> 'e' character code

The specified code in ASCII decimal is sent to the output device without conversion.

7.3 SUPERVDU Graphics Stream

7.3.0 Graphics Stream: Overview

Comprehensive facilities exist for moving the cursor around the screen. The cursor may draw a line behind itself, leaving a trail, or even define triangles which may be filled in if required. In addition, the colours used may be changed.

Note that all routines for moving the screen use the same system of coordinates as in the SUPERVDU stream. The screen can be 80, 40 or 20 units wide and 25 or 32 units high (depending on the mode) but the origin is always defined as the top left of the page (1,1).

The cursor is moved by changing the graphics coordinates. These define the screen to be 1280 pixels (i.e. picture elements) wide and 1024 pixels high. The origin in this case is defined as the bottom left of the page (0,0).

7.3.1 Current Implementation

The same restrictions on the use of pages apply as in the SUPERVDU stream in that one page and one window only are provided.

7.3.2 Graphics Stream: Function List

Available SUPERVDU functions for the Graphics Stream are given in the list below and outlined in more detail in the next section.

The graphics stream commands are alerted by:

<escape> followed by:

General

<escape>	Select Stream	
<delete>	Select debug mode	See SUPERVDU stream
'f'	Send star command	

Screen Selection

<space>	Initialise
'$'	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 mode toggle (On/Off)

7.3.3 Graphics Stream: Function Specifications

Initialise

Arguments:

<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.

Select Page

Arguments:

<escape> '$' 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:

<escape> '%' page number n

This function clears the specified page of memory. 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 0,0).

Clear Page and Select Mode

Arguments:

<escape> '&' 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
4	320 x 256	40 x 32	2
5	160 x 256	20 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.

Move Cursor Relative

Arguments:

<escape> 'h' 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:

<escape> 'i' 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:

<escape> 'j' 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 arguments would send the cursor off the screen then the command is ignored.

The function has the same effect as Plot 1, x, y.

Draw Absolute

Arguments

<escape> 'k' x, y

The cursor draws a straight line across the screen in the graphics foreground colour, moving to the absolute coordinate position (i.e. from the graphics origin) given as arguments x and y. If the arguments would send the cursor off the screen then the command is ignored.

The function has the same effect as Plot 5,x,y.

Plot

Arguments:

<escape> 'm' Plot code, x, y

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

0	Move relative to the last point
1	Draw a line relative to the last point in the graphics foreground colour
2	Draw a line relative to the last point in the logical inverse colour
3	Draw a line relative to the last point in the graphics background colour
4	Move to an absolute position
5	Draw a line from the last point to an absolute position in the graphics foreground colour
6	Draw a line from the last point to an absolute position in the logical inverse colour
7	Draw a line from the last point to the absolute position in the 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 to the specified point forming a triangle which is then filled.

All other values are reserved for future development.

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

The logical inverse colour is given by: (highest logical colour code for current mode) minus (logical colour code). Thus in a four colour mode:

logical	inverse
0	3
1	2
2	1
3	0

Define Graphics Origin

Arguments:

<escape> 'o' x, y

The graphics origin is temporarily redefined by the given x,y coordinates from the default origin of 0,0 — at the bottom left hand corner of the page.

Select Graphics Foreground Colour

Arguments:

<escape> '\' colour code, plot mode

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). The plot mode may be any of the following:

0	Plot the specified colour.
1	OR the colour with the one already there
2	AND the colour with the one already there
3	Exclusive OR the colour with the one already there
4	Invert the colour already there.

Select Graphics Background Colour

Arguments:

<escape> ']' colour code, plot mode

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:

<escape> '^' 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.

7.4 SUPERVDU Functions: Printer Stream

7.4.0 Printer Stream: General Description

The printer stream may be used to route output to the printer when it is inconvenient to use the standard 'LST:' device or when you wish to send a trace of characters to the screen. Although the screen and printer may be selected simultaneously, escape sequences will affect only one device at a time. The available features of this stream are necessarily limited by the lack of standardised printer interfaces. The printer stream is most effective when used to select Parallel/ Serial and the Baud rate for Serial operation only . Escape sequences specific to particular printers may then be sent directly through the standard printer channel.

7.4.1 Printer Stream: Function Specifications

Select Printer Route

Arguments:

<escape> 'p' route

Set printer route. The following routes may be specified:

0	Sink	all printer output is ignored.
1	Parallel	'printer' port
2	Serial	'RS423' port

Select Baud Rate

Arguments:

<escape> 'q' Baud rate

Set Baud rate for the printer. The following Baud rates may be selected:

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

Send Character Code

Arguments:

<escape> 'r' code

This function takes the decimal code (0–255) and sends it as an unconverted character. This is the only way of sending <escape> to the printer from SUPERVDU. If this sequence has to be used frequently it may be preferable to use the LST: device directly.

7.5 SUPERVDU Functions: Popular Stream

7.5.0 Popular Stream: General Description

The 'popular' terminal stream allows standard CP/M programs to be easily adapted to run on the TORCH. A selection of common control sequences are supplied which correspond closely to those used in the Televideo 912/920 terminal and the ADM31 terminal.

7.5.1 Popular Terminal: Function List

<esc> plus the following:

'+'	Clear screen
'*'	Clear screen
':'	Clear screen
';'	Clear screen
'E'	Line insert
'R'	Line delete
'T'	Clear to end of line.
't'	Clear to end of line.
'Y'	Clear to end of screen.
'y'	Clear to end of screen.
'='	Locate cursor position.
'>'	Move cursor to home position
'j'	Highlight on
'k'	Highlight off

7.5.2 Popular Terminal: Function Specifications

Clear Screen

The entire screen is cleared to null characters.

Line Insert

A blank line is inserted onto the screen at the current cursor position. The current and all subsequent lines are moved down one position.

Line Delete

The current line is deleted and all the lines below it are moved up one line.

Clear to End of Line

The current line from the cursor position to the end of the line is filled with null characters.

Clear to End of Screen

The screen is filled with null characters from the current cursor to the end of the last line of the screen.

Locate Cursor

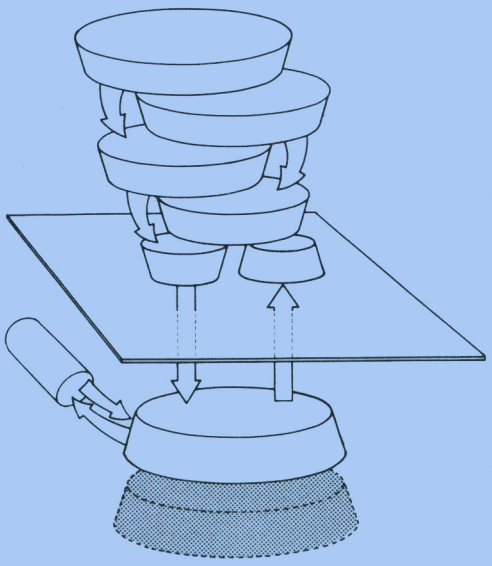
The two bytes following the '=' are taken as the row and column to which the cursor should be moved. The coordinates go from 1 to 80, 40 or 20 across and from 1 to 25 or 32 down (depending on the mode). They are offset by 31 so that coordinate position 1,1 is represented by two spaces. This is the standard cursor locating sequence used by most terminals.

Highlight On

The output will be in inverse video (that is, foreground and background colours are exchanged) until a 'highlight off' code is sent.

Highlight Off

This cancels the previous command.



The TORCH keyboard

The TORCH Keyboard

8.0 Contents

8.1 The Main Keyboard

The keys on the main keyboard are generally affected by both the shift key and the control key. Table 8.5.0 gives a key's unshifted character and code. Table 8.5.1 gives its shifted (i.e.upper case) character and code. Table 8.5.2 gives the codes produced when the key is held with control key; Table 8.5.3 gives the codes when both the shift and control keys are held down with appropriate key. All values are hexadecimal.

Note: <capitals> redefine keys a–z to be always shifted.

8.2 The Blue Function Keys

The codes produced by these keys are unaffected by the shift key. Note, however, that the values produced can be modified by Osbyte Call 225. This function sets the base number from which the function keys provide offsets (see Section 5.6.2).

8.3 The Editing Keys

Again, the codes produced by these keys are unaffected by the shift key.

The CTRL ('control') key normally has the effect of subtracting 10 hex from the value of the code generated, when depressed with the Editing keys. For example:

CTRL para	85 hex
CTRL redo	8B hex, etc.

An exception is CTRL 'underline', which generates hex 1F.

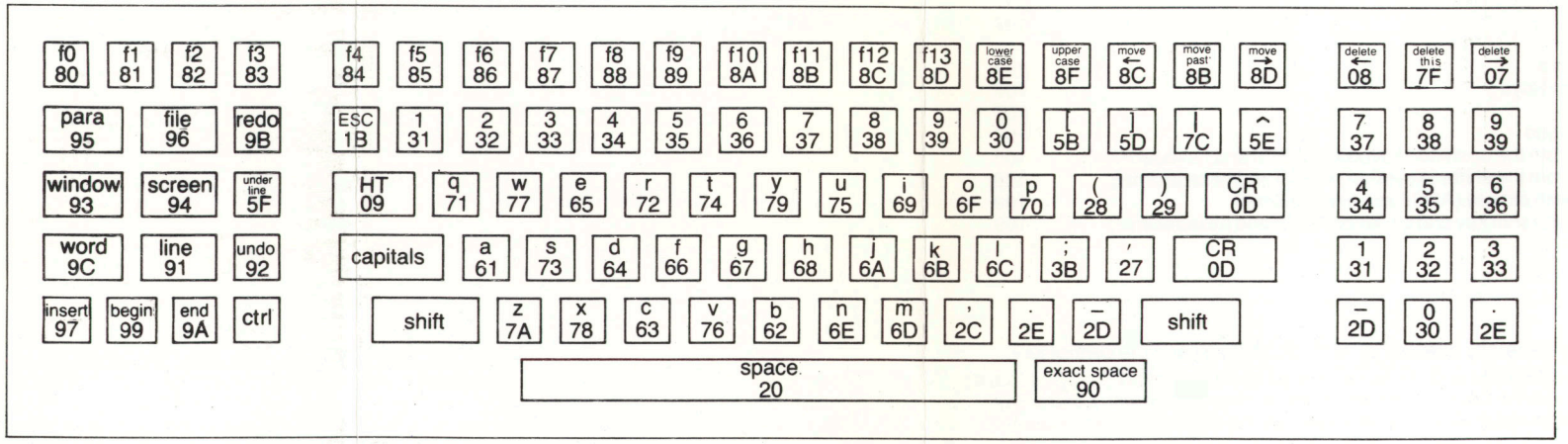
The 'exact space' key behaves like an editing key, despite its location (to the right of the 'space bar').

The key codes 80 to 9F may be modified by Osbyte Calls. See section 5.6.2 for more details.

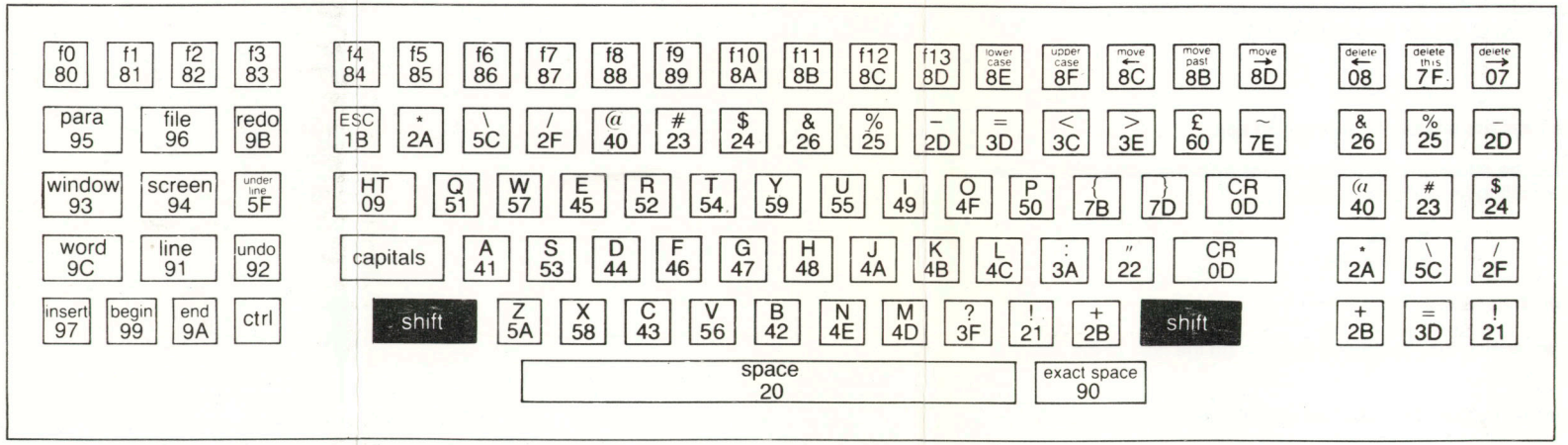
8.4 The Numeric Keypad

The numeric pad keys are functionally the same as keys 0 to 9 and the keys '–' and '.'.

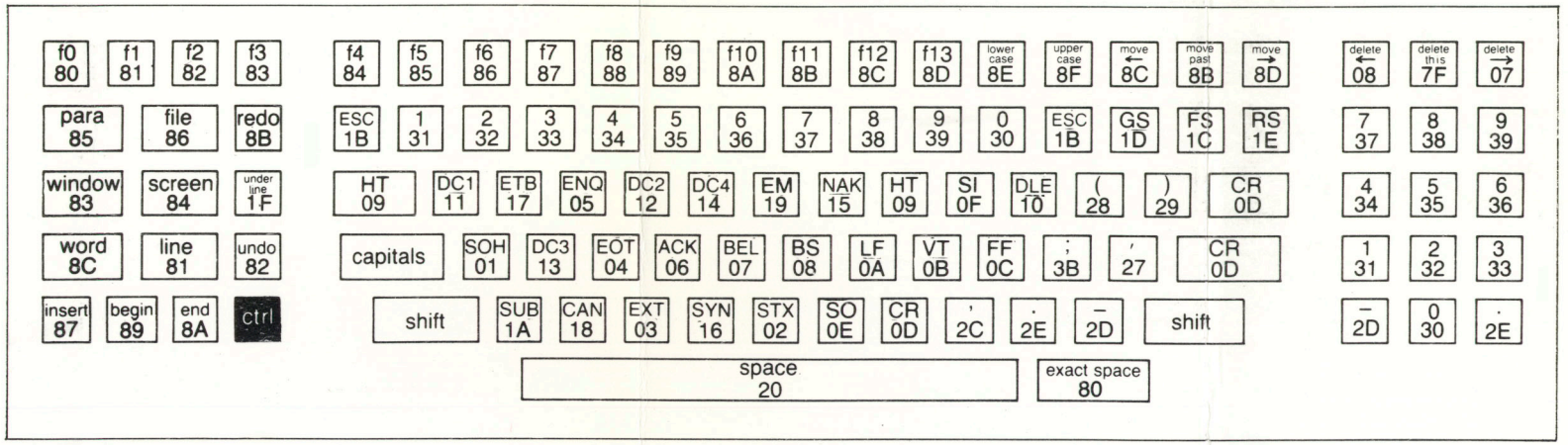
8.5.0 Codes returned from the unshifted keyboard (ASCII and hexadecimal)



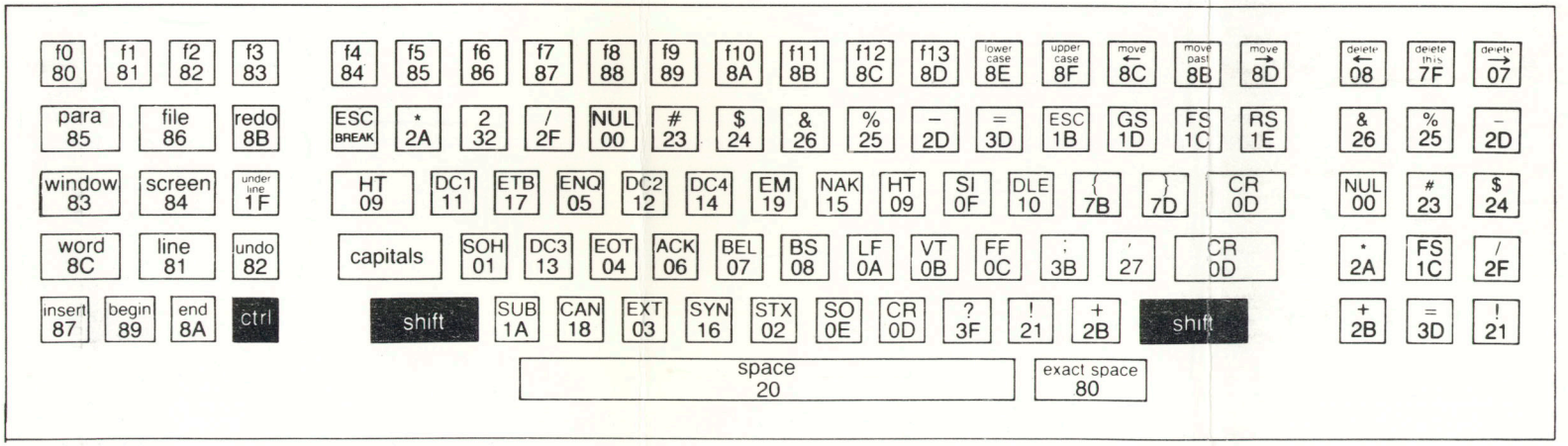
8.5.1 Codes returned from the shifted keyboard (ASCII and hexadecimal)

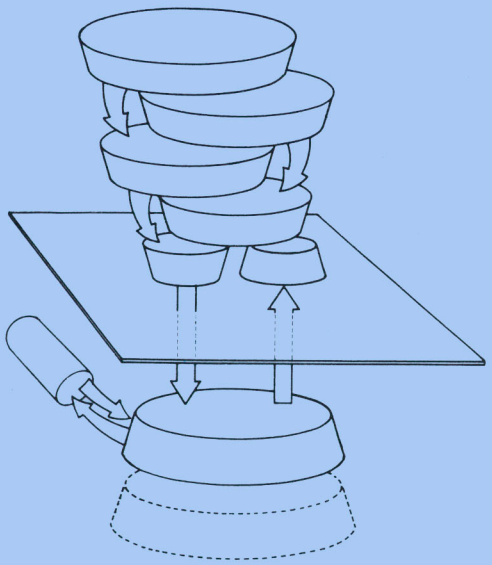


8.5.2 Codes returned with ctrl key (ASCII and hexadecimal)



8.5.3 Codes returned with shift key and ctrl keys (ASCII and hexadecimal)





Appendices

Appendices

9.0 Contents

9.1 ASCII (ISO–7) Character Set Table

HEX	MS	0	1	2	3	4	5	6	7
LS	BITS	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	£	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	EXT	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

The TORCH character codes conform completely to the ASCII (ISO–7, IA–5) standard with the exception that Accent grave (ASCII 60 hex) has been replaced by the '£' symbol.

Control characters, format effectors and delimeters are represented in the above table by the standard abbreviations:

NUL	Null
SOH	Start of heading
STX	Start of text
ETX	End of text
EOT	End of transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell
BS	Backspace
HT	Horizontal tabulation
LF	Line feed
VT	Vertical tabulation
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DLE	Data link escape
DC	Device control (DC1 to DC4)
NAK	Negative acknowledge
SYN	Synchronous idle
ETB	End of transmission block
CAN	Cancel
EM	End of medium
SUB	Substitute
ESC	Escape
FS	File separator
GS	Group separator
RS	Record separator
US	Unit separator
SP	Space (blank)
DEL	Delete

9.2 PRESTEL Character Codes Tables

9.2.0 PRESTEL C0 and GO character sets (mode 7)

bits 1–4 \ bits 5–7	0	1	2	3	4	5	6	7
0	NUL		SP	0	@	P	–	p
1		Cursor On	!	1	A	Q	a	q
2			"	2	B	R	b	r
3			£	3	C	S	c	s
4		Cursor Off	$	4	D	T	d	t
5	ENQ		%	5	E	U	e	u
6			&	6	F	V	f	v
7			'	7	G	W	g	w
8	APB		(	8	H	X	h	x
9	APF		)	9	I	Y	i	y
A	APD		*	:	J	Z	j	z
B	APU	ESC	+	;	K	←	k	¼
C	CS		,	<	L	½	l	‖
D	APR		–	=	M	→	m	¾
E		APH	.	>	N	↑	n	÷
F			/	?	O	#	o	■

key to abbreviations:

APB	Active Position Backward
APF	Active Position Forward
APD	Active Position Down
APU	Active Position Up
CS	Clear Screen
APR	Active Position Return
APH	Active Position Home

9.2.1 PRESTEL C1 display attribute control codes (produced after ESC code)

bits 1–4 \ bits 5–7	4	5
0	NUL	DLE
1	ALPHA RED	MOSAIC RED
2	ALPHA GREEN	MOSAIC GREEN
3	ALPHA YELLOW	MOSAIC YELLOW
4	ALPHA BLUE	MOSAIC BLUE
5	ALPHA MAGENTA	MOSAIC MAGENTA
6	ALPHA CYAN	MOSAIC CYAN
7	ALPHA WHITE	MOSAIC WHITE
8	FLASH	CONCEAL DISPLAY
9	STEADY	CONTIGUOUS MOSAIC
A	END BOX	SEPARATED MOSAIC
B	START BOX	
C	NORMAL HEIGHT	BLACK BACKGROUND
D	DOUBLE HEIGHT	NEW BACKGROUND
E		HOLD MOSAIC
F		RELEASE MOSAIC

9.2.2 PRESTEL Mosaic character set (produced after Graphics Select code)

bits 1–4 \ bits 5–7	2	3	6	7
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
A				
B				
C				
D				
E				
F				

9.3 Use of the speech synthesiser

The TORCH contains a TMS 5220 speech synthesis IC from Texas Instruments Inc. The programmer is advised to obtain the technical data sheets from this manufacturer or from a TORCH software supplier. These data sheets contain valuable information on the 'speech data' from which the device generates speech sounds.

The use of the speech synthesiser involves the simple algorithm given below:

Initialise the Base Processor scratchpad bytes 0 to 7 to a value of zero. Writing to the scratchpad may be carried out with User Call 14 (0E hex), passing two 1 byte arguments:

the scratchpad offset, and

the value to be written at this offset.

Pass the following values to the scratchpad area:

0 = 60 hex
1 = FF hex
2 = data 0
3 = data 1
4 = data 2
.....
.....

} speech data

(Again, writing to the Base Processor scratchpad is carried out with User Call 14 (0E hex)).

DO {

Call Osword Call 7. This is achieved by first invoking User Call 12 (0C hex), passing the Osword call number (in this case, 7) as a 1 byte argument.

Assign the following values to the scratchpad bytes:

0 = 00 hex
1 = FF hex
2 = data 0
3 = data 1
4 = data 2
.....
.....

} speech data

}

WHILE (NOT end-of-speech-data)

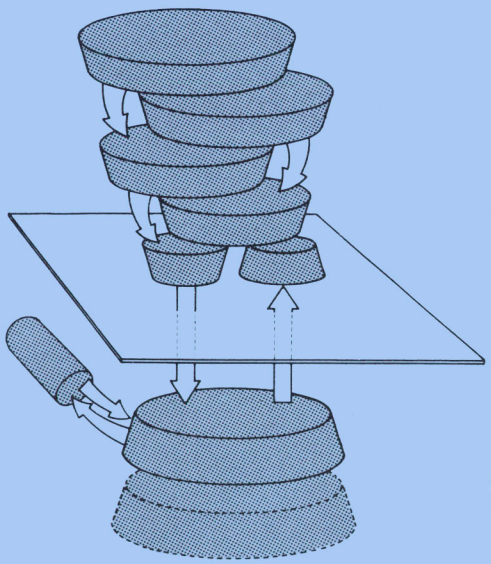
Note that to obtain realistic speech, the DO-WHILE loop in the above algorithm should be executed as fast as possible.

9.4 An example program: 'EXECUTE' (written in Z80 Assembler)

```
        .Z80
        CSEG

;*****************************************************************************
;*                                                                           *
;*        EXECUTE Command - Version 0.00                                     *
;*        Written by Duncan Booth 13th September 1982                        *
;*        Copyright  Torch Computers Ltd.                                    *
;*****************************************************************************
;*      This program is designed to allow a command (i.e. SUB) file to end   *
;* by dropping the user into an interactive program. The syntax of the       *
;* command is as follows:                                                    *
;*                                                                           *
;*        EXECUTE string_of_data_in_function_key_format.                     *
;*                                                                           *
;*        The operation of the program is:                                   *
;*                                                                           *
;*        Take the parameter string and program it into function keys 8,7,6  *
;* splitting it as necessary. If the parameter string is sufficiently short  *
;* then only as many keys as necessary are used.                             *
;*****************************************************************************

        Entry       EXECUTE

Userimm   equ       OffcOh          ;Inline user call.
Putimm    equ       Userimm + 3     ;Inline putbyte to 6502
Getbyte   equ       Userimm + 6     ;Get byte from tube to A
Putbyte   equ       Userimm + 9     ;Send byte in C to tube.


;*****************************************************************************
;*        CPN call macro                                                     *
;*****************************************************************************

CPN       MACRO     callval
          RST       30h             ;SVC restart address.
          DB        callval
          ENDM

;*****************************************************************************
;*                                                                           *
;*        This routine will access OSBYTE calls in the 6502.                 *
;*        The registers used are:   C - Osbyte call number                   *
;*                                  E - X parameter for Osbyte               *
;*                                  D - Y parameter for Osbyte               *
;*        Returned values:          E contains X result                      *
;*                                  D contains Y result                      *
;*                                                                           *
;*         Registers all saved except for DE                                 *
;*                                                                           *
;*****************************************************************************

Osbyte:   Push      BC
          Push      AF
          Call      Userimm         ;USR 15 is Osbyte
          DB        15
          Call      Putbyte         ;Send Osbyte number in C.
          LD        C,E             ;Followed by the X parameter
          Call      Putbyte         ;Send it.
          LD        C,D             ;Now the Y parameter.
          Call      Putbyte

          Call      Getbyte         ;Get the X result
          LD        E,A             ;Store it in E
          Call      Getbyte         ;Get the Y result
          LD        D,A             ;Store it in D
          Pop       AF              ;Restore registers
          Pop       BC
          RET

;*****************************************************************************
;*                                                                           *
;*         Write to scratchpad.                                              *
;*         This routine sends the byte in A to the scratchpad location       *
;*         addressed by C.                                                   *
;*         If C contains a number greater than 03FH then nothing is sent     *
;*         and the carry flag is set on return.                              *
;*         On return C is incremented to point to the next location or       *
;*         to 0 (with Z flag set) is scratchpad is full. A is set equal      *
;*         to C.                                                             *
;*                                                                           *
;*         All other registers are preserved.                                *
;*****************************************************************************
```

```
To_Scratch_Pad:
          Push      BC              ;Save registers
          LD        B,A             ;Save byte to be sent.
          Ld        A,C             ;Check range is within scratchpad.
          INC       A               ;Increment first to give next pointer.
          JR        Z,TSPend
          CP        041H            ;Check less than 41h
          CCF                       ;Carry indicates overflow of pointer.
          JR        C,TSPend        ;so abort if necessary.
          CP        040h            ;Check for end of buffer.
          JR        NZ,TSP2         ;Zero flag means we have reached the end
          XOR       A               ;So reset counter.
TSP2:     Push      AF              ;New location saved.
          Call      Userimm         ;USR 14 is write to scratchpad.
          DB        14
          Call      Putbyte         ;Send scratchpad location
          LD        C,B             ;Send data
          Call      Putbyte
          POP       AF              ;Recover new location pointer
TSPend:   POP       BC              ;Recover B register
          LD        C,A             ;Update pointer.
          RET

;*************************************************************************
;*                                                                       *
;*         Inline scratchpad routine.                                    *
;*         Sends byte following call to scratchpad location indicated by C   *
;*         Register AF corrupted, C incremented to next location.        *
;*         all others preserved.                                         *
;*************************************************************************

ISP:      EX        (SP),HL         ;HL points to parameter
          LD        A,(HL)          ;Pick up byte to be sent
          INC       HL              ;Bump return address
          EX        (SP),HL         ;Restore stack
          JR        To_Scratch_Pad  ;Then continue as above.

;*************************************************************************
;*         To access the above routine the following macro may be used.  *
;*************************************************************************

Send_Scratch        MACRO           value
          Call      ISP
          DB        value
          ENDM

;*************************************************************************
;*                                                                       *
;*         Call an Osword.                                               *
;*         This macro calls the Osword given it as a parameter.          *
;*         Registers A and C are corrupted.                              *
;*************************************************************************

Osword    MACRO     Osnumber
          Call      Userimm         ;USR 12 is Osword
          DB        12
          Call      Putimm          ;Osword number is a constant
          DB        Osnumber
          ENDM

;*************************************************************************
;*         MAIN PROGRAM                                                  *
;*************************************************************************

;*************************************************************************
;*         The following equates are used by the main program            *
;*************************************************************************

ParmStr   equ       80h             ;Parameter to command.
cr        equ       0dh
lf        equ       0ah

Execute:  LD        SP,(006h)       ;Stack for use within program.

          Call      Display
          DB        cr,lf,'EXEC command Version 0.1'
          DB        cr,lf,'14th September 1982'
          DB        cr,lf,'Copyright Torch Computers Ltd.'
          DB        0

          LD        B,8             ;Key to reprogram - start at 8 and work down as necessary.
          LD        C,0             ;Start of key buffer.
          LD        HL,ParmStr      ;Point to parameter to be programmed.
          LD        D,(HL)          ;Pick up length counter
          LD        A,D
          OR        A               ;If no parameter string
          JP        Z,Help          ;Give a help menu instead.
          INC       HL
          INC       D               ;End condition when D reaches 0.
```

```
Exec1:    Send_Scratch    '*'
          Send_Scratch    'K'         ;Key command
          Send_Scratch    'E'
          Send_Scratch    'Y'
          LD              A,B         ;Get the number of the key to reprogram
          ADD             A,'0'       ;Make into a character
          Call            To_scratch_Pad ;Put it in scratch.
          Send_Scratch    ' '         ;Separator after key number.

Tscrloop: LD              A,(HL)      ;Pick up character from buffer
          CP              '|'         ;Escaped sequences must not be split
          JR              NZ,putch    ;So put unless it is vertical bar.
          LD              A,C
          CP              3Eh         ;Last character to send must not be '|'
          LD              A,'|'       ;Restore character.
          JR              NZ,putch    ;But otherwise send as usual.
          LD              A,0Dh       ;Replace with end of line marker
          DEC             HL          ;And unread the char.
          INC             D
putch:    INC             HL          ;Bump through input buffer.
          DEC             D           ;Count length down.
          JR              Z,endparm
          Call            To_scratch_Pad
          LD              A,C         ;Check for end of scratchpad area
          CP              3fh
          JR              NZ,Tscrloop
          LD              A,0dh
          Call            To_scratch_Pad
          PUSH            BC
          PUSH            DE
          PUSH            HL
          Call            setkey      ;Setup key and put in key buffer
          POP             HL
          POP             DE
          POP             BC
          DEC             B
          JR              Exec1
setkey:   Osword          0           ;Pass command to MOS CLI
          LD              A,B         ;Key number
          ADD             A,80h       ;Make into code for that key
          LD              D,A         ;in DE for Osbyte call
          LD              E,0         ;Specify keyboard buffer
          LD              C,138       ;Osbyte 138 put character in Buffer
          Call            Osbyte
          RET
endparm:  LD              A,0dh       ;Terminate with CR
          Call            To_Scratch_Pad
          Call            Setkey      ;Setup the last key
          RST             0           ;End
Display:  EX              (SP),HL     ;Get return address
          LD              A,(HL)      ;Pick up character
          INC             HL          ;Bump return addr
          EX              (SP),HL     ;Restore stack
          OR              A           ;If char is 0
          RET             Z           ;Return
          LD              E,A         ;Character in E
          CPN             4           ;Direct console out.
          JR              Display     ;Repeat until finished.

Help:     Call            Display
          DB              0dh,0ah,0ah
          DB              cr,lf,'Use the exec command to drop out of a .SUB file into an interactive program.
          DB              cr,lf,'The syntax of this command is:'
          DB              cr,lf,'                        EXEC Sequence of commands in function key format.'
          DB              cr,lf
          DB              cr,lf,'The command line tail will be programmed into function key 8 and also 7 if'
          DB              cr,lf,'necessary. These keys will then be added to the typeahead buffer and their'
          DB              cr,lf,'values used next time keys are read.'
          DB              cr,lf,lf
          DB              cr,lf,'Example of use:'
          DB              cr,lf,'       A program called ACCOUNTS written in MBASIC is to be run immediately'
          DB              cr,lf,'   after the system is switched on. It is necessary to set up the printer'
          DB              cr,lf,'   and the screen colours beforehand.'
          DB              cr,lf,'If a file called BEGIN.SUB is created on the disc containing the following:'
          DB              cr,lf,'     ;Set up printer'
          DB              cr,lf,'     *FX 5,1'
          DB              cr,lf,'     ;Set colour'
          DB              cr,lf,'     F 2'
          DB              cr,lf,'     EXEC |XMBASIC ACCOUNTS|M'
          DB              cr,lf,'then the program may be run by simply typing CTRL+BEGIN.'
          DB              0
          RST             0

          END             Execute
```



Index

Index

10.0 Contents

10.1 Abbreviations

Note: Abbreviations have been used for the following TORCH Commands and Functions:

(CPN)	indicates a CPN function	(Section 3)
(TBC)	indicates a TORCH Base Command	(Section 4.3)
(UF)	indicates a TORCH Base User Function	(Section 4.4)

10.2 Index

Produced by Ellis Design & Advertising for Torch Computers Ltd. © 1983