REFERENCE MANUAL
PART ONE

BRITISH BROADCASTING CORPORATION
MASTER SERIES MICROCOMPUTER

The BBC Microcomputer System

Master Series

REFERENCE MANUAL – Part 1

Part number 0443,001
Issue 1, March 1986
Second revision, March 2023

Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

The product described in this manual and products for use with it, are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:

Technical Enquiries
Acorn Computers Limited
Cambridge Technopark
Newmarket Road
CAMBRIDGE CB5 8PD

All maintenance and service on the product must be carried out by Acorn Computers' authorised dealers. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of this product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any omissions from, this manual, or any incorrect use of the product.

Warning

This computer is designed to be operated with the casing as fitted at the point of manufacture. Certain sections of this manual refer to components located on the printed circuit board – the computer must be disconnected from the mains supply before (and at all times when) the top cover is removed.

This book is part of the BBC Computer Literacy Project.
Typeset by Interaction Systems Limited, Cambridge.
Cover design concept by Carrods Graphic Design.

First published 1986
Published by Acorn Computers Limited

Remastered by dv8 in 2019
stardot.org.uk/forums/viewtopic.php?t=20466

Contents

J The Disc Filing Systems (DFS and ADFS)

Index

Introduction

Your microcomputer is a sophisticated and powerful combination of hardware and software. To help you make the best possible use of all its facilities, this manual contains a description of the hardware and detailed information on using the software. Some familiarity with computer terminology and concepts is assumed so, if you are quite new to computers, DO NOT READ THIS REFERENCE MANUAL FIRST. Instead, read the Welcome Guide supplied with the computer and try out the programs on the Welcome tape or disc.

Only when you feel fairly familiar with the basic concepts and information introduced in the Welcome Guide should you look further into the material in this manual. The amount of information may appear rather daunting at first, but do not worry unduly as you are very unlikely to want, or to need to understand all of it at once. Read at your own pace, as deeply or superficially as you wish, choosing the subjects that interest you most.

The manual is divided into two parts, to make it more manageable. This volume tells you how to use the central core of the machine: the hardware and the software provided to control it. This includes information such as how to alter keyboard characteristics, produce sounds, use disc drives and connect peripherals such as printers.

Part 2 is about the software which you probably will use for most of the time. This includes information such as how to load and list a BASIC program, write assembly language routines or use the system text editor and formatter.

Each Part is divided into a number of self-contained sections, each of which deals with a single aspect of one of the major components of the system, for example, the computer's operating system. If you want details on a particular subject, you are strongly advised to use the indexes to locate entries of interest. Note that where a facility is described in both Parts 1 and 2, this volume will generally cover the more fundamental aspects and do so to a greater depth.

Note that information on the use of the VIEW word-processor and the ViewSheet spreadsheet are contained in the corresponding guides.

Conventions used in this volume

[ ]	denotes the single key on the keyboard with the corresponding legend, for example [RETURN] means the [RETURN] key.
<description>	the < and > enclose text which should not be taken literally (eg typed) but which should be interpreted according to the context in which it appears. For example, `*BUILD` <obspec> indicates that the word `*BUILD` must be followed by a sequence of characters obeying the rules defined by <obspec>. Generally, definitions of such items will be found in the introduction to the appropriate section.
[entry]	the [and] enclose an entry which is optional, the entry being either a specific sequence of characters or a description enclosed within < and > (as above). For example, `*MOUNT` [<drv>] means that either `*MOUNT` or `*MOUNT` <drv> are acceptable.
`text like this`	is used for examples of commands, programs and output that you might see on the screen.
bold text	is used for emphasis.
italic text	is used for emphasis.

A System Overview

A.1 Origins

Whilst providing considerably advanced facilities and capabilities, this microcomputer incorporates all of the features of the established BBC Microcomputer System. Established users of BBC microcomputers will, therefore, be able to employ most of their peripheral hardware, software and experience, whilst benefiting from the additional features.

The following components in this microcomputer are largely compatible with those of the BBC Model B:

– display modes 0 – 7;
– the BBC Basic interpreter;
– the machine code assembler;
– MOS commands and documented entry points;
– the Disc Filing System;
– peripheral interfacing.

In addition, display modes 128 – 135 (the shadow screen modes) are compatible with those of the BBC Model B+.

However, the following components and facilities are entirely new or significantly revised:

– more memory (RAM) available to the user (92.5 Kbytes independent of display resolution and usually independent of filing systems);
– double density disc controller together with the Advanced Disc Filing System (ADFS);
– animation facilities;
– cartridge ROM, RAM and hardware expansion capability;
– provision for internal expansions;
– expansion buffering;
– greatly extended graphics facilities;
– versatile system text editor with text formatter (EDIT);
– terminal emulation software;
– 65C12 microprocessor, with extra instructions and addressing modes;
– system calendar/clock;
– non-volatile storage of major system variables;
– sound output to domestic audio equipment;
– improved quality of black and white text on colour televisions.

A.2 Standard software

The heart of the microcomputer's software is the Machine Operating System (MOS), which controls the hardware (keyboard, display, printers etc) and provides a variety of commands to fine-tune its behaviour.

As well as the MOS, there is an extensive range of software supplied as standard, including the following applications and languages:

BASIC	–	a BBC BASIC interpreter, including a 65C12 assembler;
EDIT	–	a powerful on-screen text editor/formatter;
TERMINAL	–	full screen terminal emulation program;
VIEW	–	a flexible word processor with on-screen formatting;
ViewSheet	–	a powerful spreadsheet.

A.3 Optional software expansion

The following are among the many other languages and utilities which can be used with the computer. They are supplied in a variety of forms (ROM, cartridge, cassette, disc) and may require the connection of additional hardware such as disc drives or second processors. Consult your dealer for further details.

BCPL	–	a portable pseudocompiled language, suitable for writing compilers and system software;
COMAL	–	similar to BBC BASIC;
FORTH	–	a fast mathematically-oriented interpreted language;
LISP	–	an artificial intelligence problem-solving language;
LOGO	–	an education-oriented interpreted language with turtle graphics and list-processing;
MICRO-PROLOG	–	an artificial intelligence language;
ISO PASCAL	–	a block structured portable language;
ViewStore	–	an integrated database package which is fully compatible with both VIEW and ViewSheet.

Assembly Language development and debugging packages are also available, together with an extensive range of additional software from third-parties. Applications include:

- accountancy packages;
- educational software;
- databases;
- CAD (Computer Aided Design) packages;
- aids for the disabled;
- stock control packages;
- drawing aids;
- games.

A.4 Optional hardware expansion

A wide range of additional hardware components is available to enhance the capabilities of the system. The following peripherals for the existing BBC microcomputer system are essentially compatible with this computer:

– (external) 6502, Z80 and 32016 second processors;

– floppy disc drives;

– Winchester disc drives;

– monochrome and colour displays;

– Bitstik;

– games paddles and joysticks;

– cassette data recorders;

– IEEE adapter;

– Prestel adapters;

– Teletext adapters;

– printers.

Again, a wide variety of 'third party' peripheral units designed for use with existing BBC Microcomputer Systems is available. The following peripherals are new, designed specifically to make the best use of the additional facilities provided by this computer:

– internal Econet module;

– internal 64k 65C102 Co-processor;

– internal 512k 32016 Co-processor;

– internal 512k 80186 Co-processor.

A parallel data bus and audio connections are provided in the computer for a third-party modem, which may be fitted in the space provided inside the computer's case.

Note:

In use, the internal 65C102 Co-processor option effectively performs the same function as an externally-connected BBC Microcomputer system 6502 Second Processor, though the co-processor is much faster. Throughout this manual, reference to a 'co-processor' applies equally to a co-processor or a Second Processor.

A.5 System organisation

At its highest level, the computer is composed of system hardware and system software contained in ROMs. The system software performs three types of task, viz:

– controlling the computer hardware;
– managing information storage on external devices;
– helping the user to perform other useful tasks.

All modern microcomputers provide these facilities in some form, but in this microcomputer, they are separated physically, as follows:

The Machine Operating System (MOS)

Control of, and access to, the computer hardware is provided by the MOS. The MOS is an absolutely fundamental component of the system, providing a hardware/software interface that is used by all the other system software to access the hardware. The MOS also provides a wide variety of (software) switches which are used to alter the behaviour of the hardware and of the hardware/software interface.

Filing systems

The control of information storage on external devices is provided by Filing Systems. They provide the ability to retrieve programs or data from the filing system medium (cassette tape, floppy disc, ROM cartridge etc). You can also use most of them to store programs and data prior to loading other programs or data, or before you switch off the system. The standard filing systems are:

– the Cassette Filing System (CFS);
– the ROM (or cartridge) Filing System (RFS);
– the Disc Filing System (DFS);
– the Advanced Disc Filing System (ADFS).

The Disc Filing System (DFS) is provided for compatibility with existing BBC microcomputer systems and it is anticipated that most users will choose to make use of the more sophisticated facilities provided by the ADFS.

Whenever the computer is switched on, it has a “current filing system”, selected from one of those fitted. This is the filing system that will be used by the majority of filing system commands. The current filing system is set by the user and can be changed using certain commands.

Languages

The user requests the computer to perform tasks by giving it commands. Commands are either typed when you want them obeyed or stored in a program and executed later as a result of running the program. In either case, commands are read by the current language, which can either obey them or pass them on to be executed by the MOS or a filing system.

The current language is set by the user and can be changed using certain commands. The languages supplied as standard are:

- BBC BASIC (`*BASIC`);
- the system editor (`*EDIT`);
- the terminal emulator (`*TERMINAL`);
- the VIEW wordprocessor (`*WORD`);
- the VIEW spreadsheet (`*SHEET`).

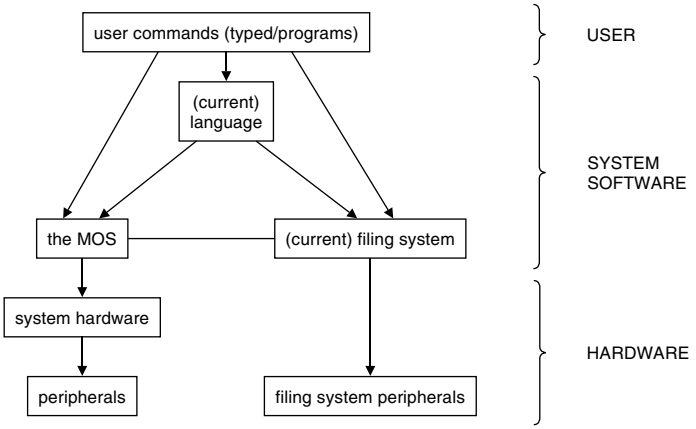
Note that a language may either be a programming language or a utility. Utilities differ from programming languages mainly by not providing the ability to run a stored sequence of commands (ie a program) and usually by having fewer data manipulation and storage commands, but resemble programming languages by obeying (typed) commands and in the way they are handled by the MOS.

The main advantages of this separation of system software are:

- the availability of multiple languages and filing systems which can be selected easily, even while a program is running;
- economy of memory usage (since part of the MOS, the languages and filing systems are all addressed in the same locations in memory);
- economy of effort and code size for system software writers.

The filing systems are carefully designed to minimise the differences between the filing devices perceived by the user, to make it easier to swap between different filing systems.

In use, the system can thus be considered as a 3-tier structure, comprising the user, the system software and the system/peripheral hardware.



(This is a considerable simplification. In particular, there is another class of system software – service ROMs – which is dealt with in greater depth in later sections.)

B The Machine Operating System (MOS)

B.1 Introduction to the MOS

In all modern computers, part of the software permanently residing in the machine is dedicated to controlling and using the hardware components. In microcomputers, this software is often combined with the language interpreter, making user access difficult. The BBC microcomputer system differs in that the software is provided as an entirely separate 35 Kbytes of ROM known as the "Machine Operating System" or MOS. The BBC BASIC interpreter leaves control of the hardware to the MOS and makes use of it through the routines provided in the MOS.

This separation of MOS and languages in this microcomputer has a number of advantages:

- in a system with more than one language, there is no unnecessary duplication of software to support the hardware;
- languages need only translate hardware-dependent commands into the corresponding MOS commands, so they are more compact and easier to write;
- the behaviour of hardware-dependent commands is consistent across languages;
- assembly language programs can use MOS routines (which will always be present, unlike language routines), so reducing the size of the code, the effort required to write it and the time taken to make it error-free;
- assembly language programs running in a second processor can use MOS routines to access locations (eg memory-mapped devices) in the I/O processor.

B.2 The role of the MOS

The MOS is a large and very complex piece of machine code software, providing support for the wide range of hardware components in the computer. Many of the sophisticated features of the computer (such as sound queuing, keyboard type-ahead, function keys, graphics primitives etc) are not features of the hardware or languages but of MOS routines that are used to interface with the hardware.

The MOS (unlike filing systems and languages which may be selected or deselected) is always there as a 'filter' lying between other software and the computer's hardware. Languages, filing systems and good assembly language programs make extensive use of MOS routines as a simpler and more efficient alternative to controlling hardware directly.

The routines in the MOS have four main types of function:

– integration of the system

  The MOS takes control when the computer is first turned on or reset; supervises the interaction of the software components of the system (filing systems and languages); services interrupts from and provides constant background processing for system and peripheral devices. Thus it functions as a sort of "glue" to make the system work as a whole.

– input and output routines

  The MOS contains routines to handle data input from, and output to, the computer's input/output (I/O) hardware. These are used by languages, filing systems and machine code programs. The main I/O hardware interfaced in this way is the keyboard, video display, printer, sound output, analogue input and RS423 input/output. Input and output through filing system hardware is largely controlled by the filing system software.

– MOS configuration

  The MOS provides a wide range of commands to alter and inspect the configuration of the hardware and the MOS routines used to service it. These commands can be used to change settings like keyboard auto-repeat rate, video interlace or current printer type. They are used by languages and filing systems but may also be executed directly in the "command mode" of languages.

– translation of filing system commands

  Many filing system commands (eg `*LOAD`, `*SAVE`, `*CAT`) that are common to a number of filing systems are translated by the MOS into a simpler form

before being passed to the current filing system for execution. This makes the filing system code more compact and allows the MOS to extend the facilities offered by existing filing systems.

Note that the MOS does not have control of the computer, except for brief periods during power-on, resets and when servicing interrupts. At other times, it is simply used by languages or applications software, as a collection of routines to give them access to system hardware.

B.3 How to use the MOS

Whenever the computer is switched on, the MOS is active, performing the integrating functions described above. The MOS is also used constantly by the current language, machine code program or filing system, to provide data input/output. Thus, in a sense you are always using the MOS whenever the system is turned on.

You can also use the MOS more deliberately to:

– change the behaviour of the system (eg alter key repeat rate, turn interlace off, select a display mode);

– obtain data from the MOS (eg the current display mode, time and date from the CMOS clock/calendar);

– input or output data (print to screen, read from the RS423 port or keyboard).

There are three basic ways of doing this:

– by executing MOS commands, either contained in a program or typed in 'command mode';

– by using MOS routines – in languages with assembler access;

– by sending VDU control codes to the screen, again, either contained in a program or typed in 'command mode'.

These three methods (executing MOS commands, using MOS routines and sending VDU control codes) are the subject of the next three sections: C, D and E.

The MOS can be used most effectively from assembly language programs or programming languages with assembly language support (eg BASIC's `CALL` statement and `USR` function). Applications programs such as the system text editor, spreadsheet or terminal emulator do not provide direct access to MOS routines. They may trap (effectively turn off) some or all of the VDU control codes and may limit the use of MOS commands although they will often provide access to these denied facilities through their own commands.

The three methods of using the MOS described above are not identical in scope. MOS commands are mainly used to alter the behaviour of MOS routines and computer hardware components, by selecting the desired configuration from a group of allowed configurations. VDU control codes are used to control the video display hardware and implement graphics and text display facilities. Direct access to MOS routines using assembly language allows you to perform

the same functions as the MOS commands and VDU control codes. You can also obtain information from the MOS, use additional MOS facilities (such as input/output) and change the behaviour of the MOS drastically by replacing many of its routines with your own.

B.4 Advantages of using the MOS

Using assembly language, it would be possible to replace virtually all the functions of the MOS with your own routines. It is even possible to implement alternatives to some MOS commands in high level languages (eg by direct access to programmable devices). Although flexibility is a major feature of this microcomputer system, this particular facility should not be used lightly. Using the MOS facilities has many advantages:

– using the MOS allows it to do its housekeeping correctly, so you will not need to understand the impact of your code on other parts of the MOS, allocate memory space for your routines to use, etc;

– using the MOS to control devices means that your code will be shorter and easier to write, the device drivers will not need debugging and you will not need such an extensive understanding of the devices being used;

– using MOS routines correctly will also help to ensure that your programs will work not just in your current configuration, but also across the Tube in a (code-compatible) co-processor, with later revisions of the MOS and with other compatible devices (eg filing systems) connected;

– using MOS commands will generally produce quicker programs than implementing the same function in a high-level interpreted language such as BASIC.

The MOS has been very carefully designed to simplify use of the computer hardware, enhance its performance through intelligent use of buffering and interrupts, and to cater for a wide variety of possible uses by providing software switches. Since the facilities of the MOS are always available to user programs, you are strongly recommended to use it wherever possible, rather than attempt to control the hardware directly.

C MOS commands

C.1 Introduction to MOS commands

MOS commands are commands of the form:

*<text> [<parameters>]

such as `*TIME`, `*CONFIGURE QUIET`, `*FX5,2` etc. They are most often used in the 'command mode' of the current language, but may also be used by `*EXEC`ing a file containing MOS commands, by running a program that contains MOS commands, or by passing the text of the command to the Operating System Command Line Interpreter (OSCLI) routine (see section D.4).

The MOS commands described in this section are those commands recognised and obeyed in full by the MOS. Other *<text> commands are obeyed by software such as filing systems or languages using the procedure described below. Such commands are described under the appropriate section title.

The main function of MOS commands is to alter the behaviour of the MOS and/or hardware components of the computer, by selecting one of a group of pre-defined settings. The MOS has been designed so that when you change one component's behaviour in this way, any MOS routines that depend upon the behaviour of that component will be adjusted accordingly.

C.2 How MOS commands are recognised

The MOS recognises two classes of command:

– commands that it executes itself;
– commands that are common to several filing systems, which it translates into the appropriate filing system call.

Commands in the second classification are referred to as filing system commands. They are interpreted in the MOS for economy of code in systems with multiple filing systems and to extend the filing systems' range of commands.

If the MOS does not recognise the command as a MOS command, it offers the command to each of the paged ROMs (filing systems, languages, service ROMs) in turn as an 'unrecognised MOS command' service call until one of them claims it. If none of the paged ROMs claims the command, the MOS finally passes the command to the current filing system via the OSFSC entry point. The filing system then either interprets it as a filing system command or generates the error message `Bad command`. If the command starts with a recognised command name it will be passed to the filing system only if the remainder of the name is alphabetic and not a legal parameter for that command.

Correct execution of MOS commands depends on the current language, which must recognise them as MOS commands and pass them unchanged to the MOS. Although commands must be spelt correctly, you can type them using any combination of capital and lower case letters (so that `*DIR`, `*Dir *diR`, for example, are exactly equivalent). You may also abbreviate commands by typing the first few characters of the command name, terminated by a full stop. Abbreviations should not be too short however, or they may be interpreted as a command other than the one intended. Where appropriate, minimum permissible abbreviations are shown for all commands described in this manual.

BBC BASIC recognises any line starting with an asterisk as a command. If a command is to be recognised by BBC BASIC, it must be at the start of a statement, although it can include leading spaces. The whole of this statement, from the initial `*` up to the carriage return, is sent to OSCLI. OSCLI recognises and attempts to execute the command at the start of the line. You must not include other commands on the same line, as they will be ignored by the

language and could cause OSCLI to reject the whole line. Thus:

```
10 IF a=1 THEN *FX4,2
```

is a valid BASIC line, but

```
10 IF a=1 THEN *FX4,2 ELSE a=2
```

and

```
10 *FX4,2:A=1
```

are not, because they include text after the command.

However, BBC BASIC also has an OSCLI statement which enables commands to be issued anywhere in a statement and, in addition, to contain variables as well as constants. For example, if variable `param%` contains 2, then the second example above could be executed by the statement:

```
10 IF a=1 THEN OSCLI"FX4,"+STR$param% ELSE a=2
```

See section K (in Part 2) for further information.

The applications programs VIEW, ViewSheet and EDIT all have command modes in which MOS commands are recognised correctly.

You can also execute commands from assembly language programs, by using the OSCLI entry point (see section D.4).

For details of how to use commands with another programming language or applications, refer to the user information for that software.

C.3 Command parameters

Command parameters are the data items included after the command name. They must be numeric or alphanumeric constants (eg `26`, `42`, `rule2`), not variables used in the current language, as OSCLI does not know anything about the language from which the command came. As described above, however, some languages provide a means of executing MOS commands with variables as parameters, such as the BASIC keyword `OSCLI`.

String constants need not be surrounded by quotation marks and may contain any combination of upper and lower case letters – they are equivalent.

Numeric constants are taken as decimal or hexadecimal depending on the command; a parameter which is normally taken as decimal may be specified as hexadecimal by preceding it with `&`. Note that the converse is not true – required hexadecimal parameters (such as memory locations) cannot be specified in decimal.

Parameters should be separated from the command and each other by one or more spaces.

C.4 MOS command summary

A complete list of MOS commands is given below in alphabetical order. The column headed **Executed by** gives an indication of the component of the system responsible for executing the command:

– commands marked MOS are related to the operation of the MOS and are described in section C.5;
– commands marked Filing System are related to Filing System operations and are described in section G.5.

The exception to this general rule is the group of `*FX` commands which are implementations of a specific MOS routine called OSBYTE; these are fully described in section D.2.

Command	Description	Executed by
`*ADFS`	Make ADFS the current Filing System	MOS
`*APPEND`	Append lines of input to a file	Filing System
`*BUILD`	Create a file from lines of input	Filing System
`*CAT`	Display disc/directory catalogue	Filing System
`*CLOSE`	Close all sequential files	Filing System
`*CODE`	Run resident machine code program	MOS
`*CONFIGURE`	Define CMOS RAM configuration settings	MOS
`*CREATE`	Create a file of given size	Filing System
`*DELETE`	Delete a named file or files	Filing System
`*DISC`	Make DFS the current Filing System	MOS
`*DUMP`	Display file in hexadecimal and ASCII	Filing System
`*EX`	Display file information	Filing System
`*EXEC`	Take subsequent input from a file	Filing System
`*FX`	Alter MOS/hardware characteristics	MOS
`*GO`	Execute machine code program	MOS
`*GOIO`	Execute machine code program in I/O processor	MOS
`*HELP`	Display ROM information	MOS
`*IGNORE`	Define printer ignore character	MOS
`*INFO`	Display file information	Filing System
`*INSERT`	Re-instate ‘unplugged’ ROM	MOS
`*KEY`	Define soft key string	MOS
`*LIBFS`	Define Library Filing System	MOS
`*LIST`	Display file in GSREAD format with line numbers	Filing System
`*LINE`	Run resident machine code program	MOS

*LOAD	Load file into memory	Filing System
*OPT	Select Filing System options	Filing System
*PRINT	Display a pure ASCII dump of a file	Filing System
*MOTOR	Switch cassette motor relay ON/OFF	MOS
*MOVE	Move file(s) between Filing Systems	Filing System
*ROM	Make ROM Filing System current	MOS
*ROMS	Display ROM contents	MOS
*RUN	Load and execute a machine code program	Filing System
*SAVE	Save block of memory as a file	Filing System
*SHADOW	Define usage of Shadow memory	MOS
*SHOW	Display soft key string allocation	MOS
*SHUT	Close all files on all active Filing Systems	Filing System(s)
*SPOOL	Send all output to a file	Filing System
*SPOOLON	Append all output to a file	Filing System
*STATUS	Display CMOS RAM configuration settings	MOS
*TAPE	Make CFS the current Filing System	MOS
*TIME	Display date and time from CMOS clock	MOS
*TV	Adjust vertical screen alignment / interlace	MOS
*TYPE	Display file in GSREAD format without line numbers	Filing System
*UNPLUG	Direct MOS to ignore specified ROM	MOS
*X	Enable access to more than one external second processor	MOS

In addition to the commands above, the MOS also recognises the commands:

```
*SRDATA
*SRREAD
*SRROM
*SRWRITE
```

which are associated with the use of the computer's sideways RAM. These commands are described in section G.7.

C.5 MOS command descriptions

The commands described below are those which are executed by the MOS itself and which relate specifically to the operation of, or to elements of the system controlled by, the MOS. They are presented in alphabetical order and the syntax definitions make use of some or all of the following definitions:

<key number> a numeric constant which identifies one of the user-programmable 'soft' keys. Normally, <key number> will be in the range 0 – 10 with 0 identifying soft key 0 and 10 identifying [BREAK]. However, the four cursor control keys and [COPY] may also be used as soft keys (see section D.2) with the following <key number> assignments:

[COPY]	11
[←]	12
[→]	13
[↓]	14
[↑]	15

<rom id> a constant in the range 0 – 15 (&00 – &0F) identifying one of the sixteen ROM sockets containing (part of) the MOS, filing systems, languages and service ROMs.

<param n> is a parameter specific to the command in question and whose definition is included in the corresponding description.

Where appropriate, the minimum permissible abbreviation is shown with each command.

*ADFS *A.

Syntax

```
*ADFS
```

Purpose

To select the Advanced Disc Filing System.

Description

`*ADFS` is used to make the Advanced Disc Filing System the current filing system (see section J).

*CODE

Syntax

`*CODE` [<param 1> [<param 2>]]

Purpose

To run a resident machine code routine.

Description

`*CODE` enters a machine code routine with A=0, X=<param 1> and Y=<param 2>. If either parameter is omitted, it is assumed to be 0.

The value of the USERV vector must be changed to point to the entry address of the routine before `*CODE` is executed (ie location &200 must contain the the LSB of the entry address and &201 the MSB), otherwise `Bad Command` will be reported.

The user routine should exit with an RTS instruction.

*CONFIGURE *CO.

Syntax

`*CONFIGURE` [<param 1> [<param 2>]]

Purpose

To define the configuration settings held in the CMOS RAM and which are made current on initial power-on and after a hard break.

`*CONFIGURE` with no parameters lists the available options.

Description

<param 1> identifies the setting to be changed, as defined below; where necessary, <param 2> defines the value to be stored in the appropriate location in the CMOS RAM. Note that the effect of `*CONFIGURE` is not immediate; to bring any change(s) into effect you must either execute a hard break or switch the computer OFF and ON again.

Parameters **Description**

`BAUD <0–8>` sets the RS423 transmit and receive rate according the specified value:

value	baud rate
0	9600
1	75
2	150
3	300
4	1200
5	2400
6	4800
7	9600
8	19200

`BOOT` reverses the actions of BREAK and SHIFT+BREAK.

`CAPS` sets the keyboard caps lock (ie *caps lock* is ON).

`DATA <0–7>` specifies the data format used by the RS423 interface according to the specified value:

value	word length	parity	stop bits
0	7	even	2
1	7	odd	2
2	7	even	1
3	7	odd	1
4	8	none	2
5	8	none	1
6	8	even	1
7	8	odd	1

`DELAY <0–255>` sets the keyboard auto-repeat delay to the specified number of hundredths of a second.

`DIR` causes ADFS to initialise with a directory selected (either the root or, if ADFS has been used since power-on, the previously selected directory).

`EXTUBE` causes an external co-processor to be used in preference to an internal co-processor (if fitted).

`FDRIVE` <0–7>

The computer is fitted with either a WD1770 or a WD1772 disc controller. `FDRIVE` defines the step time and, for ADFS, use of precompensation:

value	track to track step time		precompensation (ADFS only)
	WD1770	WD1772	
0 or 4	6ms	6ms	yes
1 or 5	6ms	6ms	no
2 or 6	30ms	3ms	yes
3 or 7	30ms	3ms	no

Most disc drives require a 6ms step time. Older drives may need the 30ms step time; modern drives may be able to operate at 3ms – consult your disc drive documentation.

`FILE` <rom id>

defines the default filing system by means of its ROM socket number.

`FLOPPY`

if the system has both floppy disc(s) and a Winchester disc connected, `*CONFIGURE FLOPPY` causes ADFS to initialise with drive 4 (floppy disc) selected.

`HARD`

if the system has both floppy disc(s) and a Winchester disc connected, `*CONFIGURE HARD` causes ADFS to initialise with drive 0 (Winchester) selected.

`IGNORE` <0–255>

defines the 'printer ignore character' (ie the character which is not to be directed to the printer), by means of its ASCII code. If the character code is omitted, all characters are sent to the printer when enabled.

`INTUBE`

causes an internal co-processor to be used in preference to an external co-processor (if fitted).

`LANG` <rom id>

defines the default language by means of its ROM socket number.

`LOUD`

sets full volume for the BELL (CTRL+G) sound.

`MODE` <mode no>

defines the default MODE setting. <mode no> may be in the range 0 – 7 or 128 – 135.

`NOBOOT`

assigns the normal functions to BREAK and SHIFT+BREAK.

`NOCAPS` resets the keyboard caps lock (ie *caps lock* is OFF).

`NODIR` causes ADFS to initialise with no directory selected (see section J.7).

`NOSCROLL` enables the scroll protect option (ie prevents hard screen scrolling if a character is placed in the last character position of the bottom screen line, depending on the current mode).

`NOTUBE` causes both internal and external co-processors to be ignored.

`PRINT <0–4>` defines the printer driver according to the value specified:

value	printer driver type
0	sink (ie no output printed)
1	'printer' port (ie parallel, Centronics type)
2	RS423 port (ie serial)
3	user printer driver
4	network printer – handled through the Econet vector

Values in the range 5 – 255 will be faulted.

`QUIET` sets half volume for the BELL ([CTRL]+G) sound.

`REPEAT <0–255>` sets the keyboard auto-repeat rate to the specified number of hundredths of a second.

`SCROLL` disables the scroll protect option (see `NOSCROLL`).

`SHCAPS` sets the keyboard [SHIFT]+[CAPS LOCK] option in which alphabetic keys produce upper case characters and SHIFTed alphabetic keys produce lower case characters.

`TUBE` check for the presence of an internal or external co-processor and, if fitted, take the action specified by either the `INTUBE` or `EXTUBE` option.

`TV <0–255> [<0–1>]` alter the vertical screen alignment and set or reset the interlace option.

The first parameter is interpreted as a signed byte ie 1 means a movement of one line up and 255 (−1) means a movement of one line down. If the first

parameter is 0 no adjustment is made to the vertical screen alignment.

If the second parameter is 0 the screen interlace option is switched ON; if it is 1, the screen interlace is switched OFF. If the second parameter is omitted the value zero is assumed.

*DISC *DI.

Syntax

`*DISC`

Purpose

To select the 1770 Disc Filing System.

Description

`*DISC` is used to make the Disc Filing System the current Filing System (see section J)

*FX

Syntax

`*FX` <param 1> [<param 2> [<param 3>]]

Purpose

To alter MOS/hardware characteristics.

Description

Each `*FX` command is an implementation of a call to the MOS OSBYTE routine using the parameters specified (see section D.2).

*GO

Syntax

`*GO` [<param 1>]

Purpose

To execute an assembly language program in a single processor system or in the language processor of a system with an internal or external co-processor.

Description

If <param 1> is specified, `*GO` takes <param 1> to be the (hexadecimal) address of the program's entry point and starts execution of the routine. If <param 1> is omitted, `*GO` selects OSCLI as the current language.

*GOIO

Syntax

`*GOIO` <param 1>

Purpose

To execute an assembly language program in the I/O processor of a system equipped with an internal or external co-processor (ie across the Tube).

Description

`*GOIO` takes <param 1> as the (hexadecimal) address of the program's entry point and starts execution of the routine. `Bad address` is reported if <param 1> is missing or invalid.

*HELP *H.

Syntax

`*HELP` [<param 1>]

Purpose

To display information about the MOS, Filing Systems and Languages available.

Description

If <param 1> is omitted, `*HELP` displays the title string and a list of keywords (possibly none) for each available ROM. If <param 1> is specified, each ROM in turn attempts to match the string with one of its keywords and, if a match is found, the corresponding expansion is displayed.

*IGNORE *IG.

Syntax

`*IGNORE` [<param 1>]

Purpose

To set the 'printer ignore character'.

Description

It is sometimes necessary to prevent certain characters from being sent to the printer. This facility is most commonly used to 'filter out' line feed characters (ASCII 10) sent to printers which perform an automatic line feed (LF) when sent a carriage return. <param 1> is the ASCII code of the character to be ignored; if it is omitted, all characters are sent to the printer.

*INSERT *INS.

Syntax

*INSERT <rom id>

Purpose

To re-instate an 'unplugged' ROM (see *UNPLUG below).

Description

*INSERT informs the MOS that the ROM in socket <rom id> is to be made available for selection. Note that the *ROMS command (see below) will show the effect of any *INSERT commands immediately although the ROM itself will not be made available until after the next hard reset.

*KEY *K.

Syntax

*KEY <key number> [<text>]

Purpose

To assign a string to one of the user-programmable soft keys.

Description

The MOS allows you to define strings of text (each of up to 255 characters in length) to the available soft keys. <text> (which may be the 'null string') is composed of ASCII characters obtained using the following conventions:

ASCII code	symbols used
0	¦@
1 – 26	¦<letter> ie ¦A (or ¦a) = ASCII 1, ¦M (or ¦m) = ASCII 13 etc
27	¦[
28	¦\
29	¦]
30	¦^ or ¦~
31	¦_ or ¦£
32 – 126	keyboard character, except for " and ¦ which are denoted by ¦" and ¦¦ respectively
127	¦?
128 – 255	¦!<coded symbol> ie ASCII 128 = ¦!¦@ ASCII 129 = ¦!¦A etc

To include leading spaces in a definition, you must enclose the string in quotes

(") which are not included in the definition. To include a single " character in the string, use ¦".

Soft key definitions are normally unaffected by a soft break but are lost following a hard break or if BREAK is pressed while the soft key buffer is being changed.

*LIBFS

Syntax

*-<filing system name>-LIBFS

Purpose

To define the 'Library' Filing System.

Description

Whilst *LIBFS is a MOS command, its syntax and use are closely linked with the reference material associated with Filing Systems. A full description of this command may therefore be found in section G.

*LINE

Syntax

*LINE <param 1>

Purpose

To run a resident machine code routine.

Description

<param 1> is a string of characters which must be terminated by a carriage return character (ASCII 13).

*LINE enters a machine code routine with A=1, X=<LSB of the string address> and Y=<MSB of the string address>.

The value of the USERV vector must be changed to point to the entry address of the routine before *LINE is executed (ie location &200 must contain the the LSB of the entry address and &201 the MSB), otherwise Bad Command will be reported.

The user routine should exit with an RTS instruction.

*MOTOR *M.

Syntax

`*MOTOR <0–1>`

Purpose

To switch the cassette motor relay ON or OFF.

Description

`*MOTOR 0` switches the cassette motor relay OFF; `*MOTOR 1` switches the cassette motor relay ON.

*ROM

Syntax

`*ROM`

Purpose

To select the ROM Filing System.

Description

`*ROM` is used to make the ROM Filing System the current Filing System.

*ROMS

Syntax

`*ROMS`

Purpose

To display a list of the paged ROMs occupying the available ROM sockets.

Description

`*ROMS` produces a list of the sixteen ROM positions together with the title string and primary version number of the ROM occupying each socket. The display format for each socket is:

`ROM` <rom id in hex.> <ROM title or ?> [`unplugged`]

The appearance of a `?` in the ROM title field indicates that the socket is either empty or that it contains a ROM whose header information does not conform to that required for a paged ROM.

If the word `unplugged` appears at the end of a line it indicates that the paged ROM in question has been the subject of a `*UNPLUG` command (see below). Note however, that ROMs so marked will still be available for selection if the computer has not been reset.

*SHADOW *SH.

Syntax

`*SHADOW` [<0–255>]

Purpose

To specify usage of Shadow memory.

Description

There are two areas of memory which can be selected for use as display memory:

– from &3000 (variable) to &7FFF in the main memory map;

– a similar area outside the main memory map, known as the Shadow RAM.

Use of the first area produces a display mapping compatible with the BBC Model B microcomputer; use of the second leaves more memory available for the user (as in the BBC Model B+). By default, display modes 0 – 7 use the area reserved in the main memory map and modes 128 – 135 use the shadow RAM.

`*SHADOW 0`	(or simply `*SHADOW`) selects 'shadow mode' in which the MOS uses shadow RAM for the screen display on subsequent mode changes, regardless of the mode number used.
`*SHADOW` <1–255>	switches shadow mode OFF – the MOS will select either main or shadow memory according to the mode number used in all subsequent mode changes.

*SHOW

Syntax

`*SHOW` <key number>

Purpose

To display the string assigned to a user-programmable soft key.

Description

`*SHOW` displays the string assigned to a soft key using `*KEY` (above). The string itself will be enclosed in quotes (") even if they were omitted from the original definition.

*STATUS *ST.

Syntax

`*STATUS` [<param 1>]

Purpose

To display CMOS RAM configuration setting(s).

Description

If <param 1> is omitted, `*STATUS` displays a full list of the default values held in the CMOS RAM. IF <param 1> is present, it must be one of the settings shown under `*CONFIGURE` (above) and only the corresponding value is displayed.

Note that changes to the settings held in the CMOS RAM are implemented only after a power-on or hard break. `*STATUS` displays the settings which will come into effect when the machine is next switched on or subjected to CTRL+BREAK (ie recent changes may be 'pending').

*TAPE *T.

Syntax

`*TAPE` or
`*TAPE3` or
`*TAPE12`

Purpose

To select the Cassette Filing System (and baud rate).

Description

`*TAPE` or `*TAPE12` makes the Cassette Filing System the current Filing System and sets the default transfer rate of 1200 baud.

`*TAPE3` is identical in effect except that the transfer rate is set to 300 baud – provided for increased reliability when using poor quality cassette recorders / tape.

*TIME *TI.

Syntax

`*TIME`

Purpose

To display the date and time from the CMOS RAM.

Description

`*TIME` produces a standard display containing the day, date and time. The format is:

<day>`,`<date>`.`<time>

where <day> is a three character abbreviation for the day of the week;
<date> is of the form *dd mmm yyyy*, *dd* being the day number, *mmm* being an abbreviation for the month name and *yyyy* being the year
<time> is of the form *hh*`:`*mm*`:`*ss*, the hours being specified in 24hr clock notation.

*TV

Syntax

`*TV` <0–255> [<0–255>]

Purpose

To specify the vertical screen alignment and interlace option.

Description

The first parameter is interpreted as a signed byte ie 1 means a vertical adjustment of one line up, 255 (−1) means a vertical adjustment of one line down.

The second parameter controls the use of the screen interlace: 0 switches the interlace ON, 1 switches the interlace OFF. If the second parameter is omitted the current interlace setting is left unchanged.

*UNPLUG *UNP.

Syntax

`*UNPLUG` <rom id>

Purpose

To direct the MOS to ignore a specified ROM.

Description

By default, all ROMs resident in the machine are candidates for selection or service calls and, in addition, they may use up RAM for their own private workspace. It is therefore possible that certain `*` commands are intercepted by the wrong ROM or that the amount of main memory is reduced because the

'hidden RAM' (see section F) is insufficient for the workspace requirements of all active Filing Systems/languages.

*UNPLUG enables you to overcome problems of this type by making the MOS 'pretend' that a ROM has been removed from its socket. The single parameter is the ROM socket number which can be ascertained from the display produced by *ROMS (above).

The effect of a *UNPLUG command comes into effect only after the next power-on or hard break although any intervening *ROMS command will show the specified ROM as unplugged.

*X

Syntax

*X

Purpose

To enable access to more than one external second processor.

Description

This command accesses hardware location &FEE8 (see section F) and then waits for a reset, at which point another external second processor may be selected (if suitable hardware is fitted).

C.6 Error messages

The MOS generates the following error messages in response to invalid commands.

`Bad command` (Error 254)

The command is not recognised by the MOS or any of the paged ROMs and is not a file which the current filing system can `*RUN` or `*EXEC`.

This error is also generated if you issue a `*CONFIGURE` command with an illegal parameter.

`Bad key` (Error 251)

The key number in a `*KEY` command is illegal (ie outside the range 0–15).

`Bad name` (Error 204)

The command is either in excess of 12 characters in length or contains illegal characters.

`Bad string` (Error 253)

The string parameter associated with the command is invalid. The most common cause of this error is the failure to match " characters, especially with `*KEY`.

`I cannot run this code` (Error 0)

An attempt has been made to select a paged ROM as a language when its header indicates that it is designed for use with a different processor (eg a Z80 second processor).

`Key in use` (Error 250)

This error is generated if an attempt is made to program a soft key to re-program itself, for example:

```
*KEY0*KEY0*STATUS¦M*TIME¦M
```

Note that it is permissible to program a soft key to re-program a different soft key.

`This is not a language` (Error 0)

This message indicates an attempt to select a paged ROM as a language when its header information indicates that it is not a language.

D Using MOS routines

D.1 Introduction to MOS routines

Programs written in assembly language do not have access to the sophisticated input and output commands available in higher level languages such as BASIC or Pascal. In many microcomputers, the assembly language programmer is forced to provide these functions by writing his or her own code but with this computer the programmer is able to use the MOS routines described in this chapter.

Using the MOS routines is strongly recommended, as it has many advantages:

– ease of programming

  The MOS routines are efficient, documented and well-proven; using them will reduce the effort required to produce a reliable, working program.

– compactness of code

  Code to use existing MOS routines is usually more concise than code to replace it and hence will run faster. Use of user code will use up RAM/ROM space.

– independence of hardware revisions

  MOS routines are designed to provide an interface to the hardware, which is consistent (where possible) in later versions of the computer.

– preserving the integrity of the MOS and MOS variables

  MOS routines can be relied upon whereas user code may have unforeseen side-effects.

The documented MOS routines are listed overleaf. The routine names are not known by the MOS or assembler; they are used solely for simplifying documentation.

Routine name	Address	Function
OSBYTE	&FFF4	Perform MOS command requiring byte parameters
OSWORD	&FFF1	Perform MOS command requiring parameter block
OSCLI	&FFF7	Interpret *<text> command
OSRDCH	&FFE0	Read character from input stream
OSRDSC	&FFB9	Read byte from screen memory or paged ROM
OSWRCH	&FFEE	Write character to output stream(s)
OSNEWL	&FFE7	Write LF+CR to output stream
OSASCI	&FFE3	Write character to output stream(s), or LF+CR if character is CR
OSWRSC	&FFB3	Write a byte to screen
OSFIND	&FFCE	Open or close a file
OSFILE	&FFDD	Load or save a file
OSARGS	&FFDA	Load or save data about a file
OSGBPB	&FFD1	Load or save a part of a file
OSBPUT	&FFD4	Save a byte to a file
OSBGET	&FFD7	Load a byte from a file
OSEVEN	&FFBF	Generate an event
GSINIT	&FFC2	Initialise GSREAD string
GSREAD	&FFC5	Read a byte from string

MOS routines are executed by means of a technique referred to as indirection whereby the actual addresses of the code to be executed are held in RAM in memory locations called vectors. The addresses given in the table above merely contain instructions which cause control to be passed to the address in the corresponding entry in the vector. The vectors are initialised on reset.

The use of indirection means that programmers can rely on the fact that the call addresses will always be the same in different versions of the MOS although the routines themselves may occupy different locations. Direct access to a MOS routine (through its address in the indirection vector) should therefore be considered to be a very risky procedure.

The use of indirection also means that the user is able to intercept calls to MOS routines by inserting the address of a user routine in the appropriate vector.

D.2 Executing MOS commands (OSBYTE)

OSBYTE is the routine used to read and set MOS variables and to invoke many of the available operating system facilities. One important example of its use is for executing the *FX commands mentioned in section C.5.

In fact, *FX commands and OSBYTE calls are (nearly) one and the same; the former being the more common implementation in the 'command mode' of the current language and the latter in assembly language programs. The most significant difference between the two is the ability of some OSBYTE calls to read and return values, as will be seen in the following sections.

Call address &FFF4
Indirected via &20A (BYTEV)

Parameters are passed to OSBYTE using the 65C12 accumulator (A), and the X and Y registers:

– the accumulator contains the call number;
– the X register may contain a parameter;
– the Y register may contain a parameter.

using any of the methods below.

– In the command mode of the current language, OSBYTE calls are implemented by means of *FX commands, ie:

  *FX <call number>,<parameter 1>,<parameter 2>

  Such commands may only write MOS data.

  The parameters should be separated from the command and each other by one or more spaces or commas. A parameter is taken to be zero if is is expected but omitted.

– In assembly language, an OSBYTE call takes the form:

```
LDA #<call number>
LDX #<parameter 1>
LDY #<parameter 2>
JSR &FFF4
```

  Such calls may either read, write, or read/write MOS data and, where appropriate, values are returned in the X and Y registers.

– In the BASIC language, OSBYTE calls may be specified either as `*FX` commands (for writing only), or using the `USR` function:

```
A%=<call number>
X%=<parameter 1>
Y%=<parameter 2>
Z%=USR(&FFF4)
```

After executing these statements, `Z%` will be a number representing the contents of the status register, the Y register, the X register and the accumulator on exit from the OSBYTE routine. Thus if `Z%=&12345678`:

the status register contained &12
the Y register contained &34
the X register contained &56
the accumulator contained &78.

OSBYTE parameter specification

As described above, the accumulator must always contain the required call number when OSBYTE is entered. However, the choice of values in the X and Y registers depends upon the purpose of the call.

A large number of these calls are for reading and writing certain MOS variables and, when using these calls:

– to read a value, the X register should contain 0 and the Y register should contain 255 (&FF);

– to write or read/write a value, the X register should contain the value and the Y register should contain 0.

– to set or clear individual bits using OSBYTE, the Y register is used as a mask and the X register contains the required settings:

action on bit	**bit setting in X**	**bit setting in Y**
set to 0	0	0
set to 1	1	0
nothing	0	1

So, for example, if:

X= 0 0 0 0 0 0 1 1 (&03) and
Y= 1 1 1 1 0 0 0 0 (&F0)

the effect is to set bits 0 and 1 (to 1), reset bits 2 and 3 (to 0) and leave bits 4,5,6 and 7 unchanged.

The read and read/write functions behave as they do because the corresponding calls perform the operation:

<new value> = (<old value> AND <Y>) EOR <X>

In the other OSBYTE calls, the meaning of X and Y depends upon the particular call.

OSBYTE always preserves the accumulator.

D.2.1 OSBYTE call summary

A full list of OSBYTE call numbers is given below, in 'functional' groups; the availability of read, write or read/write functions is indicated in the command description.

Descriptions of the OSBYTE calls are given in section D.2.2; those marked □ are either new calls or calls which have a new function in this computer.

Analogue to digital conversion

The analogue to digital converter (ADC) converts voltages (analogue data) connected to the analogue input into a digital equivalent (a number) that can then be manipulated by programs. Although the ADC can only convert a single analogue input at a time, the MOS allows you to read up to four channels. It does this by switching the ADC to sample each channel in turn, waiting for the ADC interrupt which signifies that conversion is complete for that channel, then reading and storing the data for that channel before switching to the next. If all four channels are selected, they are sampled in the order 4, 3, 2, 1, 4, 3, 2, 1 etc.

When you 'read' an analogue channel using the appropriate MOS routine, you are actually reading the result (stored by the MOS) of converting the last sample from the channel in question. These readings have a range of 12 bits and you can rely upon 10 bits (provided that you use a low-noise voltage source). Sampling and conversion takes approximately 10 milliseconds per channel. You can select a faster (4 millisecond), 8-bit resolution mode if required.

OSBYTE calls relating to the operation of the analogue to digital converter are:

dec.	hex.	description
16	10	Write number of ADC channels
17	11	Write next ADC channel to be sampled
128	80	Read ADC channel/get buffer status
188	BC	Read current ADC channel
189	BD	Read maximum ADC channel number
190	BE	Read/write ADC conversion type

BREAK and associated effects

The BREAK key is handled by the MOS in quite a different way from the rest of the keys on the keyboard. It has two main effects:

– it applies a reset signal to the microprocessor and many of the other major chips;
– it causes the MOS to execute a routine to service the BREAK (eg by re-initialising chips).

BREAK on its own generates a 'soft break' which reinitialises the current language and filing system.

SHIFT+BREAK also generates a 'soft break' and usually boots the current filing system (but see `*CONFIGURE BOOT`).

CTRL+BREAK generates a 'hard break' which selects the default filing system and language, clears all the buffers and resets the machine to the state it is normally in after initial power-on.

OSBYTE calls relating to the effect of the BREAK key are:

dec.	hex.	description
200	C8	Read/write BREAK and ESCAPE effect
247	F7	Intercept BREAK vector
248	F8	
249	F9	
253	FD	Read last BREAK type
255	FF	Read/write start-up options

CMOS RAM

The computer has 50 bytes of battery-backed CMOS RAM that can be used to store information which can be changed easily but that is not lost when the computer is turned off. The CMOS RAM usage is:

bytes	usage
0–19	20 bytes used by the MOS and other system software to store defaults such as the printer driver type, keyboard repeat rate, editor display mode etc.
20–29	reserved for future expansion
30–39	reserved for use by commercial third parties
40–49	available for use by the user

The CMOS RAM also maintains the computer's clock and calendar (see `*TIME`).

OSBYTE calls are provided to both read from and write to locations in the CMOS RAM.

dec.	hex.	description
161 □	A1	Read from CMOS RAM
162 □	A2	Write to CMOS RAM

Econet

After adding an internal Econet module and the Advanced Network Filing System (ANFS) ROM to the computer, it can be used as a station in an Acorn Econet network.

Three OSBYTE calls relate to the use of Econet. Whilst brief details are provided in the following OSBYTE call descriptions, further information will be found in the Advanced Econet User Guide.

dec.	hex.	description
206	CE	Read/write Econet OS call interpretation status
207	CF	Read/write Econet input interpretation status
208	D0	Read/write Econet output interpretation status

ESCAPE and escape effects

ESCAPE (ASCII 27) is referred to as the 'escape character' and its depression causes the MOS to set the top bit of location &00FF, thereby generating an 'escape condition'. The escape character and the actions to be taken on the occurrence of an escape condition are set by default but may be changed by the user.

The generation of escape conditions from RS423 is normally disabled but may be enabled if required (see under Serial (RS423) interface, below).

The OSBYTE calls associated with ESCAPE and escape effects are:

dec.	hex.	description
124	7C	Clear escape condition
125	7D	Set escape condition
126	7E	Acknowledge escape condition with side effects
220	DC	Read/write ESCAPE character
229	E5	Read/write ESCAPE key status
230	E6	Read/write ESCAPE effects

Filing System operations

The OSBYTE calls listed below are associated with Filing System operations. Some are directly equivalent to MOS commands described in section C.5, others provide a means of accessing, and changing, variables maintained by the MOS on behalf of the current filing system.

dec.	hex.	description
109	6D	Make temporary filing system permanent
119	77	Close all `*SPOOL`/`*SPOOLON` or `*EXEC` files
127	7F	Check for end-of-file on an opened file
137	89	Switch cassette motor relay
139	8B	Write filing system options
140	8C	Select Cassette Filing System
141	8D	Select ROM Filing System
164	A4	Check processor type
176	B0	Read/write CFS timeout counter
183	B7	Read/write Cassette/ROM filing system switch
198	C6	Read/write `*EXEC` file handle
199	C7	Read/write `*SPOOL` file handle

Input and output

The OSBYTE calls below are associated with general input/output operations – ie those which apply to more than one I/O route. Calls specific to a single input or output (ie ADC, keyboard, printer RS423 etc) are described under the appropriate heading. I/O via Filing Systems is described in sections G – J.

Characters are input (read) from two main sources:

– the keyboard;

– RS423 port;

using the Operating System Read Character routine (OSRDCH) – see section D.5.

Characters are output (written) to one or more of four main destinations:

– the screen;

– parallel printers (via the printer port);

– serial devices (via the RS423 port);

– a spool file (via some filing system);

using the Operating System Write Character routine (OSWRCH) – see section D.7.

The MOS uses dedicated areas of memory (buffers) to store data input from the keyboard and RS423 interface and to queue data for output to the RS423 interface, printer and sound synthesiser. Buffers increase the efficiency of the system by reducing the amount of waiting that data-providing and data-removing routines have to do. The various buffers are addressed using buffer numbers which are given in the full description of the appropriate calls.

OSBYTE calls associated with general-purpose input-output operations are:

dec.	hex.	description
2	02	Specify input stream
3	03	Specify output streams
15	0F	Flush buffer
21	15	Flush selected buffer
138	8A	Insert character code into buffer
145	91	Get character from buffer
152	98	Examine buffer status
153	99	Insert character code into buffer checking for ESCAPE
177	B1	Read input source
236	EC	Read character destination status

Interpretation of characters

Characters are stored in input buffers as single bytes, with values ranging from 0 – 255. These values may be placed in the buffer in a number of ways:

– from the keyboard;
– from the RS423 input (if selected for input);
– directly (see OSBYTE calls 138 and 153).

The way in which characters are subsequently interpreted is set by default but may be changed using the following OSBYTE calls:

dec.	hex.	description
4	04	Enable/disable cursor editing
219	DB	Read/write [TAB] key code
221	DD	Read/write interpretation of input values 192–207
222	DE	Read/write interpretation of input values 208–223
223	DF	Read/write interpretation of input values 224–239
224	E0	Read/write interpretation of input values 240–255
225	E1	Read/write soft key interpretation
226	E2	Read/write [SHIFT]+soft key interpretation
227	E3	Read/write [CTRL]+soft key interpretation
228	E4	Read/write [SHIFT]+[CTRL]+soft key interpretation

237		ED	Read/write cursor editing status
238	□	EE	Read/write numeric keypad interpretation
254	□	FE	Set effect of SHIFT on numeric keypad

Interrupts and events

An interrupt is a hardware signal which makes the processor stop what it is doing and execute a particular MOS routine. The routine (referred to as an 'interrupt service routine') determines the source of the interrupt, performs the appropriate actions then returns control to the task that was interrupted.

Interrupts are generated by a variety of devices. For example, when a key is pressed on the keyboard, the keyboard circuitry generates an interrupt which causes the MOS to scan the keyboard matrix to detect which key has been pressed, store this information and set a flag so that the other keyboard input routines are activated.

The processor used in this computer recognises two types of interrupt:

– Non-maskable interrupts (NMIs)

  which are generated by the disc controller and the Econet interface;

– Interrupt requests (IRQs)

  which are used by devices such as the keyboard, ADC, RS423 and cassette interfaces.

An NMI always causes the MOS to execute its NMI service routine but the processor can be instructed to 'ignore' IRQs by setting its interrupt disable flag with an SEI instruction (see section P).

The MOS makes extensive use of interrupts to increase its flexibility and speed. Before you interfere with the interrupt system, you must have a good understanding of the system and of assembly language programming. Facilities are provided for you to handle given interrupts yourself or to disable them entirely.

A safer alternative to changing interrupt handling is to use the OSBYTE calls below which are associated with the handling of 'events' – important changes detected by the MOS during normal interrupt handling. If the MOS detects an event, it checks whether that event has been enabled by the user and, if it has, jumps to the user-supplied event-handling code pointed to by the EVNTV vector (&0220).

Each event is assigned a number which is given in the full description of each corresponding call.

dec.	hex.	description
13	0D	Disable event
14	0E	Enable event
231	E7	Read/write IRQ bit mask for user 6522
232	E8	Read/write IRQ bit mask for 6850 (RS423)
233	E9	Read/write interrupt bit mask for system 6522

The keyboard

The keyboard is the main source of user input. When a character is typed, the code it represents is placed in the keyboard input buffer from where it is accessed by OSRDCH. The actual procedure is fairly complex but it is summarised below.

Pressing a key that produces a character (ie not [CAPS LOCK], [SHIFT LOCK], [SHIFT] or [CTRL]) generates an IRQ that calls the keyboard interrupt service routine. This routine then:

– scans the keyboard matrix to read which key(s) are pressed;
– stores the 'internal key number' of the of the key(s) as the 'current keys pressed information';
– sets the 'keydown flag', indicating that a key has been pressed;
– initialises the 'keydown counter'.

The keydown flag is checked every 10 milliseconds during the regular housekeeping performed by the MOS. When the flag is set, the MOS checks the 'current keys pressed information' to see which key is meant to be pressed and scans the keyboard to see if it is still pressed.

If the current key is still pressed, the MOS decrements the keydown counter to see if it is time to insert the corresponding code into the keyboard buffer. When the counter reaches zero the MOS checks if either [SHIFT] or [CTRL] are depressed, checks the state of both shift lock and caps lock and translates the internal key number into the appropriate character code. At this stage, the code is placed in the keyboard buffer and the keydown counter is reset.

If the key has been released, the MOS changes the current keys pressed information accordingly and resets the keydown flag.

The OSBYTE calls below are associated with keyboard input characteristics:

dec.	hex.	description
11	0B	Write keyboard auto-repeat delay
12	0C	Write keyboard auto-repeat rate
18	12	Reset soft keys

118	76	Reflect keyboard status in keyboard LEDs
120	78	Write keys pressed information
121	79	Keyboard scan
122	7A	Keyboard scan from 16 decimal
129	81	Read key with time limit
172	AC	Read address of keyboard translation table
178	B2	Read/write keyboard semaphore
196	C4	Read keyboard auto-repeat delay
197	C5	Read keyboard auto-repeat rate
201	C9	Read/write keyboard status
202	CA	Read/write keyboard status byte
216	D8	Read/write length of soft key string

Memory-mapped I/O

The main system components are controlled and used by reading and writing addresses in the range &FC00 – &FEFF in the I/O processor. For documentation purposes, these pages of memory are assigned names:

Page	Address range	Name
&FC	&FC00–&FCFF	FRED
&FD	&FD00–&FDFF	JIM
&FE	&FE00–&FEFF	SHEILA

The use of these locations is described in section F. However, by using the OSBYTE calls below you can make sure that your programs will work with a co-processor.

dec.	hex.	description
146	92	Read from I/O area FRED
147	93	Write to I/O area FRED
148	94	Read from I/O area JIM
149	95	Write to I/O area JIM
150	96	Read from I/O area SHEILA
151	97	Write to I/O area SHEILA

Miscellaneous functions

The OSBYTE calls below are those which provide functions which do not fit sensibly into any other category.

dec.		hex.	description
0		00	Display MOS version
1		01	Write user flag
107	☐	6B	Write 1MHz bus selection status
108	☐	6C	Write usage of main/shadow memory
130		82	Read machine high order address
131		83	Read Operating System High Water Mark (OSHWM)
132	☐	84	Read top of user RAM
133	☐	85	Read top of user RAM for given mode
136		88	Execute user code
164	☐	A4	Read processor type
166		A6	Read start address of MOS variables
180		B4	Read/write Operating System High Water Mark (OSHWM)
215		D7	Read/write start-up message suppression status
239	☐	EF	Read/write `*SHADOW` state
240	☐	F0	Read country flag
241		F1	Read/write user flag
243		F3	Read timer switch state
244		F4	Read/write soft key consistency flag

Paged ROMs

Paged ROMs are ROM or EPROM chips inserted into particular sockets on the computer's printed circuit board. In the standard machine, they are used to hold all the standard system software other than the MOS.

Paged ROMs may also be used to hold other languages, utilities, filing system software and data and may be plugged directly into the main board (as for standard system software) or mounted in ROM cartridges for insertion into the cartridge ROM sockets.

All ROMs are addressed in the same memory range ie &8000–&BFFF.

When the computer is turned on or subjected to a hard break, the MOS scans the paged ROMs fitted, stores information about them (the ROM types) and attempts to select a current language and filing system.

Code in paged ROMs is executed under the control of the MOS, immediately after one of the following:

– the MOS issues a service call;

  the MOS issues service calls for a variety of reasons, such as when the computer is reset, when it executes a BRK instruction or when it receives a MOS command that it does not recognise;

– a call is made which indirects through an extended vector;

  calls of this nature are normally used in response to filing system commands, to access routines in paged filing system ROMs. However, the user can also change MOS vectors to extended vectors so that calls through that vector are passed to a paged ROM.

– a call is made to a language entry point;

  the MOS calls the language entry point as the last action in selecting a language, as a result of a soft or hard break, a language selection command (eg `*EDIT`) or execution of an OSBYTE 142 (see below).

Note that as far as the MOS is concerned, a paged ROM will only exist if it has not been 'unplugged' (see `*UNPLUG`).

OSBYTE calls associated with paged ROM operations are:

dec.	hex.	description
22 □	16	Increment ROM polling semaphore
23 □	17	Decrement ROM polling semaphore
142	8E	Enter language ROM
143	8F	Issue paged ROM service request
168	A8	Read address of ROM pointer table
170	AA	Read address of ROM information table
179 □	B3	Read/write ROM polling semaphore
186	BA	Read ROM number active on last BRK
187	BB	Read ROM number of socket containing BASIC
252	FC	Read/write current language ROM number

Printers

The vast majority of printers use one of two standard hardware interfaces:

– Centronics compatible (parallel);
– RS423/RS232 compatible (serial).

The computer provides both interfaces, together with the software routines required to use printers attached to them.

Printed text is output via OSWRCH after enabling printer output with OSBYTE 3 (see above, under 'Input and output') and/or sending ASCII 2 to the VDU driver (see section E).

The OSBTYE calls directly concerned with printer handling are:

dec.	hex.	description
5	05	Write printer driver type
6	06	Write printer ignore character
123	7B	Inform MOS of printer driver going dormant
182 □	B6	Read/write NOIGNORE state
245	F5	Read printer driver type
246	F6	Read/write printer ignore character

Serial (RS423) interface

The serial interface allows you to send data to and receive data from devices equipped with an RS423 interface. It is mainly used for terminal emulation, and for connecting serial printers. Note that RS423 is fully compatible with the specification for RS232.

dec.	hex.	description
7	07	Write RS423 receive rate
8	08	Write RS423 transmit rate
156	9C	Write serial ACIA control register and copy
181	B5	Read/write RS423 input interpretation status
191	BF	Read/write RS423 busy flag
192	C0	Read serial ACIA control register copy
203	CB	Read/write RS423 input buffer minimum space
204	CC	Read/write RS423 input ignore flag
205	CD	Read/write RS423 destination
242	F2	Read copy of serial processor ULA register

Sideways RAM

Two calls are provided for determining information relating to the 64K of sideways RAM fitted in the computer.

dec.	hex.	description
68	44	Test RAM presence
69	45	Test pseudo/absolute use of bank

Sound

The sound synthesiser is mainly controlled using OSWORD calls (see section D.3) because of the number of parameters involved. However, the following OSBYTE calls are available:

dec.	hex.	description
210	D2	Read/write sound suppression status
211	D3	Read/write BELL channel
212	D4	Read/write BELL sound information
213	D5	Read/write BELL frequency
214	D6	Read/write BELL duration

The Tube

The Tube is a high-speed asynchronous interface between the microcomputer and a co-processor. The Tube and its operating system function almost automatically and are designed to be 'transparent' so that programs that work in the standard computer will also work correctly with, for example, a 65C102 co-processor. Note, however, that this will be the case only if you use the MOS commands, routines and techniques described in this manual.

dec.	hex.	description
157	9D	Write byte across Tube
234	EA	Read flag indicating Tube presence

Video display

The video display is mainly used and controlled by sending ASCII codes to the character output routine (OSWRCH) under the control of the VDU driver. The VDU driver is described in section E, but the following OSBYTE calls are available:

dec.	hex.	description
9	09	Write duration of first colour
10	0A	Write duration of second colour
19	13	Wait for vertical sync
20 ☐	14	Restore default font definitions
25 ☐	19	Restore a group of font definitions
112 ☐	70	Select main/shadow memory for VDU driver access
113 ☐	71	Select main/shadow memory for display
114 ☐	72	Write usage of shadow memory
117	75	Read VDU status

134		86	Read text cursor position
135		87	Read character at text cursor position/screen mode
144		90	Set vertical screen shift and interlace option
154		9A	Write video ULA control register and copy
155		9B	Write video ULA palette register and copy
160		A0	Read VDU variable value
165		A5	Read output cursor position
174		AE	Read VDU variables origin
184		B8	Read OS copy of video ULA control register
185		B9	Read OS copy of video ULA palette register
193		C1	Read/write flash counter
194		C2	Read/write count for duration of first colour
195		C3	Read/write count for duration of second colour
217		D9	Read/write paged mode line count
218		DA	Read/write bytes in VDU queue
250	☐	FA	Read memory accessed VDU driver
251	☐	FB	Read memory displayed

D.2.2 OSBYTE call descriptions

Each OSBYTE call is described below, the sequence being numerical order of call number. Refer to the functional listing in section D.2.1 for information on the calls related to specific topics.

The information provided for each call is presented as follows:

– *Call number and function*

  The call number is specified in decimal (and hexadecimal); the function is a brief statement which indicates the availability of read, write and read/write functions;

– *Description*

  A description of the effect of the call is given if necessary;

– *Entry parameters*

  Listed under 'Entry parameters' are the permissible contents of the X and Y registers – the accumulator must always contain the call number;

  Assembly language programmers will load the appropriate values explicitly before calling &FFF4;

  BASIC programmers calling OSBYTE via the `USR` function will load the appropriate values into `X%`, `Y%` and `A%` before evaluating `USR(&FFF4)`;

– **FX equivalent*

  If it is appropriate to issue the OSBYTE call as a `*FX` command, the syntax of the command is given under this heading;

– *Conditions on exit*

  This entry defines the content of the registers on exit from the call. Note that this information is irrelevant if the call is implemented by means of its `*FX` equivalent because there is nowhere for the returned values to be stored.

If necessary, any additional information is provided under the heading *Notes*.

OSBYTE 0 (&00) Display MOS version

OSBYTE 0 has the effect of performing a BRK instruction and displaying the MOS version number.

Entry parameters : X=0 executes a BRK and displays the OS version
X=1 executes an RTS and returns the Operating system version

***FX equivalent** : `*FX0`

On exit : X=<OS version>

OSBYTE 1 (&01) Write user flag

This call provides a means of writing data to the user flag (location &281). The flag is zero by default

Entry parameters : X=<new value>

***FX equivalent** : `*FX1,`<0–255>

On exit : X=<old value>

Note

This call uses OSBYTE 241/&F1 and is left free for user applications.

OSBYTE 2 (&02) Specify input stream

Input may be taken from either the keyboard (by default) or the RS423 port. This call specifies the selection for all subsequent input.

Entry parameters : X=0 selects keyboard input and disables RS423
X=1 selects and enables RS423 input
X=2 selects keyboard input and enables RS423

***FX equivalent** : `*FX2,`<0–2>

On exit : X=0 indicates previous input was from the keyboard
X=1 indicates previous input was from RS423
Y is undefined

OSBYTE 3 (&03) Specify output stream

Output may be directed to one or more of the display, printer port, RS423 port and a file opened by `*SPOOL`. This call specifies the selection for subsequent output.

Entry parameters : X determines the output stream(s) according to its bit settings:

bit	effect if set
0	Enables RS423 driver
1	Disables VDU driver
2	Disables printer driver
3	Enables printer (independent of CTRL+B or CTRL+C)
4	Disables spooled output
5	Not used
6	Disables printer driver (unless the character is preceded by a VDU 1 or equivalent)
7	Not used

***FX equivalent** : `*FX3,`<0–127>

On exit : X=<old stream specification>
Y is undefined

OSBYTE 4 (&04) Enable/disable cursor editing

This command defines the effect of the four cursor control keys and COPY.

Entry parameters : X=0 enables cursor editing
X=1 disables cursor editing and assigns ASCII codes:

key	ASCII code
COPY	135
←	136
→	137
↓	138
↑	139

X=2 disables cursor editing and assigns soft key numbers:

key	soft key number
COPY	11
←	12
→	13
↓	14
↑	15

***FX equivalent** : `*FX4,`<0–2>

On exit : X=<old effect>
Y is undefined

OSBYTE 5 (&05) Write printer driver type

This call selects the printer driver type (and hence the printer port) for subsequent printer output.

Entry parameters : X=0 selects printer sink (printer output ignored)
X=1 selects parallel (Centronics) printer driver
X=2 selects RS423 output (see note)
X=3 selects user printer driver
X=4 selects network printer driver

***FX equivalent** : `*FX5,`<0–4>

On exit : X=<old driver type>
Y is undefined

Notes

1. If X=2 on entry and RS423 output has been enabled using OSBYTE 3, this call will act as a printer sink.
2. Interrupts are enabled by this call.
3. Output stream selections are not reset by a soft break.
4. This call waits for the printer buffer to empty before selecting the new printer driver.

OSBYTE 6 (&06) Write printer ignore character

This call sets the 'printer ignore character' (ie the character which is not to be sent to the printer unless preceded by ASCII 1). It is almost equivalent in effect to `*IGNORE` (see note).

Entry parameters : X contains the character code to be ignored

***FX equivalent** : `*FX6,`<0–255>

On exit : X=<old ignore character>
Y is undefined

Note

`*FX6,`*n* is exactly equivalent to `*IGNORE`*n*. `*FX6` (with no parameter) sets the printer ignore character to ASCII 0, whereas `*IGNORE` (with no parameter) does not set a printer ignore character ie all characters are sent to the printer.

OSBYTE 7 (&07) Write RS423 receive rate

This call sets the RS423 baud rate for receiving data. The actual format of the data is set using OSBYTE 156/&9C (see below).

Entry parameters : X=0 selects 9600 baud
X=1 selects 75 baud
X=2 selects 150 baud
X=3 selects 300 baud
X=4 selects 1200 baud
X=5 selects 2400 baud
X=6 selects 4800 baud
X=7 selects 9600 baud
X=8 selects 19200 baud

***FX equivalent** : `*FX7,`<0–8>

On exit : X=Y=<old serial ACIA control register contents>

OSBYTE 8 (&08) Write RS423 transmit rate

This call sets the RS423 baud rate for transmitting data. The actual format of the data is set using OSBYTE 156/&9C (see below).

Entry parameters : as for OSBYTE 7

***FX equivalent** : `*FX8,`<0–8>

On exit : as for OSBYTE 7

OSBYTE 9 (&09) Write duration of first colour

Actual colours 8 to 15 are displayed as a sequence of two alternating colours. By default, each colour is displayed for 0.5 seconds at a time. This command allows you to alter the duration for which the first colour is displayed.

Entry parameters : X=0 sets an infinite duration and forces the mark state
X=n sets the duration to n vsync units (ie fiftieths of a second in the UK)

***FX equivalent** : `*FX9,`<0–255>

On exit : X returns the old setting
Y is undefined

OSBYTE 10 (&0A) Write duration of second colour

As for OSBYTE 9, but with reference to the second colour.

Entry parameters : as for OSBYTE 9

***FX equivalent** : `*FX10,`<0–255>

On exit : as for OSBYTE 9

OSBYTE 11 (&0B) Write keyboard auto-repeat delay

Each key on the keyboard must be held down for a number of hundredths of a second (50 by default, or as reset by `*CONFIGURE DELAY`) before it will begin to auto-repeat. This call enables this initial delay to be reset (until the next break or power-off).

Entry parameters : X=0 disables auto-repeat
X=n sets the auto-repeat delay to n hundredths of a second

***FX equivalent** : `*FX11,`<0–255>

On exit : X=<old delay>
Y is undefined

OSBYTE 12 (&0C) Write keyboard auto-repeat rate

Unless auto-repeat has been disabled, each key on the keyboard will auto-repeat (after the auto-repeat delay has elapsed, see above) at either the default rate of 8 hundredths of a second or at the rate specified by `*CONFIGURE REPEAT`. This call enables the auto-repeat rate to be reset (until the next hard break or power off).

Entry parameters : X=0 resets the auto-repeat delay and rate to their default settings
X=n sets the auto-repeat rate to n hundredths of a second

***FX equivalent** : `*FX12,`<0–255>

On exit : X=<old rate>
Y is undefined

OSBYTE 13 (&0D) Disable event

All events are assigned a unique number and this call provides a means of disabling specific events.

Entry parameters : X=<event number> as given below:

event number	event
0	output buffer empty
1	input buffer full
2	character entering buffer
3	ADC conversion complete
4	start of vertical sync
5	interval timer crossing zero
6	escape pressed
7	RS423 error
8	network error
9	user

***FX equivalent** : `*FX13,`<0–9>

On exit : X=Y=<old enable state> (0=disabled)

OSBYTE 14 (&0E) Enable event

This call provides a means of enabling specific events.

Entry parameters : as for OSBYTE 13/&0D

***FX equivalent** : `*FX14,`<0–9>

On exit : X=Y=<old enable state> (>0=enabled)

OSBYTE 15 (&0F) Flush buffer

This call is a general purpose input buffer flush which can be applied either to all buffers or just the current input buffer. See also OSBYTE 21/&15.

Entry parameters : X=0 flushes all buffers
X=1 flushes the current input buffer

***FX equivalent** : `*FX15,`<0–1>

On exit : X and Y are undefined

Note

The current buffer content is discarded.

OSBYTE 16 (&10) Write number of ADC channels

By default, each of the four ADC channels is sampled and converted in turn so that each reading is updated every 40 milliseconds. This call enables the number of channels to be changed so that if, for example, only two channels are required, each will be updated every 20 milliseconds.

Entry parameters : X=0 disables ADC sampling
X=n selects n channels

***FX equivalent** : `*FX16,`<0–4>

On exit : X=<old number of channels>

Notes

1. If n exceeds four, four is assumed.
2. Switching off all sampling marginally increases the processor's speed.

OSBYTE 17 (&11) Write next ADC channel to be sampled

This call aborts the current ADC conversion and starts sampling again from the channel specified, ie sampling and conversion then continues down to channel 1 in the normal manner.

Entry parameters : X=<ADC channel number>

***FX equivalent** : `*FX17,`<1–4>

On exit : X is preserved (or reset to 4 – see note under OSBYTE 16/&10)
Y is undefined

Note

OSBYTE 17 can be used to take a single sample from a channel which has been disabled (using OSBYTE 16/&10).

OSBYTE 18 (&12) Reset soft keys

Soft key assignments are normally maintained other than after a hard reset or power off. This call allows the soft key buffer to be cleared if necessary.

Entry parameters : none

***FX equivalent** : `*FX18`

On exit : X is undefined
Y is preserved

OSBYTE 19 (&13) Wait for vertical sync

This call is used to synchronise a program with the signal produced at the start of each video frame (which is 50 times per second in the UK). It is particularly useful in animation routines.

Entry parameters : none

***FX equivalent** : `*FX19`

On exit : X and Y are undefined

OSBYTE 20 (&14) Restore default font definitions

The shape of the character displayed when printing ASCII codes 32 – 255 may be redefined using the VDU driver (see section E). Any such changes remain in force until the computer is subjected to a hard break or power off and this command may be used to restore the default character definitions for ASCII codes in the range 32 – 126 (see OSBYTE 25/&19 for redefining the remaining characters).

Entry parameters : none

***FX equivalent** : `*FX20`

On exit : X and Y are undefined

Note

On the BBC Model B microcomputer, OSBYTE/*FX 20 was used to explode the soft character RAM allocation, according to the value of the parameter supplied. Software designed to run on both the Master Series and BBC Model B microcomputers should use this call with X=6 (ie `*FX20,6`) . This parameter will be ignored in Master Series computers and will fully explode the font in the BBC Model B.

OSBYTE 21 (&15) Flush selected buffer

Whereas OSBYTE 15/&0F may be used to flush all buffers (or the current input buffer), OSBYTE 21 is used to flush an individual buffer, specified by its buffer number.

Entry parameters : X=<buffer number> as given below:

buffer number	buffer
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

***FX equivalent** : `*FX21,`<0–7>

On exit : X is preserved
Y is undefined

OSBYTE 22 (&16) Increment ROM polling semaphore

This call increments the semaphore which, when non-zero, makes the MOS issue a ROM service call (reason code &15) at centisecond intervals. See section F.7 for further information.

Entry parameters : none

***FX equivalent** : `*FX22`

On exit : X is preserved
Y is undefined

OSBYTE 23 (&17) Decrement ROM polling semaphore

See also OSBYTE 22/&16.

Entry parameters : none

***FX equivalent** : `*FX23`

On exit : X is preserved
Y is undefined

OSBYTE 24 (&18) is reserved.

OSBYTE 25 (&19) Restore a group of font definitions

Entry parameters : X=<group identifier> as given below:

identifier	group of character codes
0	32 – 255
1	32 – 63
2	64 – 95
3	96 – 127
4	128 – 159
5	160 – 191
6	192 – 223
7	224 – 255

***FX equivalent** : `*FX25,`<group identifier>

On exit : X and Y are undefined

OSBYTE 26 (&1A) – 67 (&43) are currently unused.

OSBYTE 68 (&44) Test RAM presence

This call enables the presence or absence of each of the four banks of sideways RAM to be tested.

Entry parameters : none

***FX equivalent** : none

On exit : X contains a byte in which the least significant four bits indicate the presence or absence of a 16K bank of sideways RAM:

bit	bank
0	4
1	5
2	6
3	7

Bits set indicate the presence of the corresponding bank; bits clear indicate their absence.

Note

In the standard machine, this call will always return the value &0F, indicating the presence of all four banks of sideways RAM. Changes to the PCB links (see section F.5) will, however, affect this value.

OSBYTE 69 (&45) Test pseudo/absolute use of bank

This call enables the current use of each of the four banks of sideways RAM to be determined.

Entry parameters : none

***FX equivalent** : none

On exit : X contains a byte in which the least significant four bits indicate the current use (ie pseudo or absolute addressing) of a 16K bank of sideways RAM:

bit	**bank**
0	4
1	5
2	6
3	7

Bits set indicate the use of pseudo addressing; bits clear indicate the use of absolute addressing

Note

The value returned for this call will be zero on power-up but may change depending upon the use of `*SRROM` and `*SRDATA` commands (see section G.7).

OSBYTE 70 (&45) – 106 (&6A) are currently unused.

OSBYTE 107 (&6B) Write 1MHz bus selection status

The computer provides two groups of signal and data lines referred to as the '1MHz buses'. These can be used to connect a variety of devices to the computer, such as a Teletext adapter or music synthesiser.

The external '1MHz bus' runs at 1Mhz and is accessed via the external 1MHz bus connector.

The internal '1MHz bus' runs at 2MHz and is accessed via the cartridge interface. This is a new facility which should be used for devices which are able to run at the full 2MHz clock speed used by the computer.

Only one of the two buses may be selected at a time and this call provides a means of selecting or switching between the two.

Entry parameters : X=0 selects the external bus
X=1 selects the internal bus

***FX equivalent** : `*FX107,`<0–1>

On exit : X is preserved
Y is undefined

OSBYTE 108 (&6C) Write usage of main/shadow memory

The shadow memory is a 20K section of RAM which can be paged into memory locations &3000–&7FFF, displacing the main memory which would normally be addressed there. Shadow memory is used to implement the 'shadow display modes' in which it serves as the display RAM, allowing the user to use the whole of RAM from OSHWM to &7FFF.

This call provides one means of paging the shadow RAM into and out of the main map.

Entry parameters : X=0 pages main memory into the main map (immediate)
X=1 pages shadow memory into the main map (immediate)

***FX equivalent** : `*FX108,`<0–1>

On exit : X is preserved
Y is undefined

OSBYTE 109 (&6D) Make temporary Filing System permanent

Section G.1 describes a means of including a filing system name in various commands, ie it is possible to instruct a filing system other than the current filing system to carry out some specified action. For example:

```
*-DISC-CAT
```

would cause the Disc Filing System to display a catalogue of the current directory regardless of the current filing system. This call has the effect of making the temporary filing system (DFS in the example) the current filing system.

Entry parameters : none

***FX equivalent** : none

On exit : X and Y are undefined

OSBYTE 110 (&6E) and 111 (&6F) are currently unused.

OSBYTE 112 (&70) Select main/shadow memory for VDU access

This call specifies which of main or shadow memory will be updated by the VDU driver (regardless of the memory currently being displayed) until the next mode change. Used in conjunction with OSBYTE 113 (below) this facility allows rapid screen switching such as might be required in animation routines.

Entry parameters : X=0 sets writing to main or shadow memory according to the mode in use (immediate)
X=1 sets writing to the memory in the main map (immediate)
X=2 sets writing to the shadow memory (immediate)

***FX equivalent** : `*FX112,`<0–2>

On exit : X=<old setting>
Y is undefined

OSBYTE 113 (&71) Select main/shadow memory for display

This call specifies which of main or shadow memory will be displayed (regardless of the memory currently being updated by the VDU driver) until the next mode change.

Entry parameters : X=0 sets display to main or shadow memory depending upon the mode in use (immediate)
X=1 sets display to memory in the main map (immediate)
X=2 sets display to shadow memory (immediate)

***FX equivalent** : `*FX113,`<0–2>

On exit : X=<old setting>
Y is undefined

OSBYTE 114 (&72) Write usage of shadow memory

Equivalent to `*SHADOW`

Entry parameters : X=0 selects shadow memory on subsequent mode changes, regardless of the mode selected

X=<1–255> selects main or shadow memory on subsequent mode changes depending on the mode selected

***FX equivalent** : `*FX114,`<0–255>

On exit : X=<old usage>
Y is undefined

OSBYTE 115 (&73) – 116 (&74) are currently unused.

OSBYTE 117 (&75) Read VDU status

This call returns the contents of the VDU status byte.

Entry parameters : none

***FX equivalent** : none

On exit : X=<VDU status> with set bits indicating the following:

bit	status when set
0	Printer output enabled by VDU2
1	Scrolling disabled
2	Paged scrolling selected
3	Text window in force
4	Currently in a shadow mode
5	Printing at graphics cursor enabled by VDU5
6	Cursor editing in progress
7	Screen disabled with VDU21

Y is preserved

OSBYTE 118 (&76) Reflect keyboard status in keyboard LEDs

The settings of caps lock and shift lock are held in a location referred to as the keyboard status byte and, under normal circumstances, will be shown by the keyboard LEDs. However, it is possible to write to the keyboard status byte directly (see OSBYTE 202) although the LEDs will not be changed accordingly.

This call ensures that the current contents of the keyboard status byte are reflected in the LEDs.

Entry parameters : none

***FX equivalent** : `*FX118`

On exit : X is preserved
Y is undefined

OSBYTE 119 (&77) Close all *SPOOL/*SPOOLON or *EXEC files

Entry parameters : none

***FX equivalent** : `*FX119`

On exit : X and Y are undefined

OSBYTE 120 (&78) Write keys pressed information

When a key is pressed, the MOS stores a number representing it in location &00EC of the I/O processor. If another key is depressed before the first is released, the first number is transferred to &00ED and a number representing the second keystroke is placed in &00EC. These locations are referred to as the keys pressed information and the numbers used are the internal key numbers (see table below) plus 128.

This call enables you to mimic the depression of keys from software by writing internal key numbers to the keys pressed locations.

Entry parameters : X=<first internal key number> (see table)
Y=<second internal key number> (see table)

***FX equivalent** : `*FX120,`<first internal key number>[`,`<second internal key number>]

On exit : X and Y are preserved

Notes

1. OSBYTE 120 will not normally cause keystrokes to be placed in the input buffer because it does not set the keydown flag – see OSBYTE 138/&8A.
2. The SHIFT and CTRL keys are not stored as keys pressed information but merely change the bit settings in the keyboard status byte – see OSBYTE 202/&CA.

Key	internal key number		Key	internal key number	
SHIFT	&00	0	0	&36	54
CTRL	&01	1	P	&37	55
Q	&10	16	[	&38	56
3	&11	17	↑	&39	57
4	&12	18	*keypad* +	&3A	58
5	&13	19	*keypad* −	&3B	59
f4	&14	20	*keypad* RETURN	&3C	60
8	&15	21	CAPS LOCK	&40	64
f7	&16	22	A	&41	65
−	&17	23	X	&42	66
^	&18	24	F	&43	67
←	&19	25	Y	&44	68
keypad 6	&1A	26	J	&45	69
keypad 7	&1B	27	K	&46	70
f0	&20	32	@	&47	71
W	&21	33	:	&48	72
E	&22	34	RETURN	&49	73
T	&23	35	*keypad* /	&4A	74
7	&24	36	*keypad* DELETE	&4B	75
I	&25	37	SHIFT LOCK	&50	80
9	&26	38	S	&51	81
0	&27	39	C	&52	82
_	&28	40	G	&53	83
↓	&29	41	H	&54	84
keypad 8	&2A	42	N	&55	85
keypad 9	&2B	43	L	&56	86
1	&30	48	;	&57	87
2	&31	49	]	&58	88
D	&32	50	DELETE	&59	89
R	&33	51	*keypad* #	&5A	90
6	&34	52	*keypad* *	&5B	91
U	&35	53	*keypad* ,	&5C	92

Key	internal key number		Key	internal key number	
TAB	&60	96	ESCAPE	&70	112
Z	&61	97	f1	&71	113
SPACE	&62	98	f2	&72	114
V	&63	99	f3	&73	115
B	&64	100	f5	&74	116
M	&65	101	f6	&75	117
,	&66	102	f8	&76	118
.	&67	103	f9	&77	119
/	&68	104	\	&78	120
COPY	&69	105	→	&79	121
keypad 0	&6A	106	*keypad* 4	&7A	122
keypad 1	&6B	107	*keypad* 5	&7B	123
keypad 3	&6C	108	*keypad* 2	&7C	124

OSBYTE 121 (&79) Keyboard scan

This call provides a means of checking whether a particular key or one of a range of keys is depressed.

Entry parameters : X=<internal key number EOR &80> (for a single key)
X=<lowest internal key number> (for a range of keys)
Y=0

***FX equivalent** : none

On exit : *for a single key:*
X<0 (ie top bit set) indicates that the key is depressed

for a range of keys:
X=255 indicates that no key in the given range is depressed otherwise
X=<internal key number of key depressed>

Notes

1. If two (or more) keys are depressed when checking for a range of key depressions, the lowest internal key number is returned in X.
2. Any key depressions are entered into the input buffer during execution of this call.
3. As with OSBYTE 120/&78, the SHIFT and CTRL keys may not be scanned with this call.

OSBYTE 122 (&7A) Keyboard scan from 16 decimal

Equivalent to OSBYTE 121/&79 with X=16.

Entry parameters : none

***FX equivalent** : none

On exit : as for OSBYTE 121/&79 with a range of keys

OSBYTE 123 (&7B) Inform MOS of printer driver going dormant

This call is only of interest to users writing their own (user) printer drivers.

Entry parameters : X=3 (ie printer buffer number)

***FX equivalent** : none

On exit : X and Y are preserved

OSBYTE 124 (&7C) Clear escape condition

This call clears any escape condition without further action other than informing the Tube, if active.

Entry parameters : none

***FX equivalent** : `*FX124`

On exit : X and Y are preserved

OSBYTE 125 (&7D) Set escape condition

This call simulates depression of the current escape character. The Tube is informed, if active, although an escape event is not generated.

Entry parameters : none

***FX equivalent** : `*FX125`

On exit : X and Y are preserved

OSBYTE 126 (&7E) Acknowledge escape condition

This call attempts to clear an escape condition.

Entry parameters : none

***FX equivalent** : none

On exit : X=255 indicates that the escape condition has been cleared
X=0 indicates that the escape condition has not been cleared

Note

The actions necessary to clear an escape condition will depend upon whether escape condition side effects have been enabled or disabled – see OSBYTE 230.

OSBYTE 127 (&7F) Check for end of file on an opened file

This call provides a means of ascertaining whether the end of an open file has been reached.

Entry parameters : X=<file handle>
Y=0

***FX equivalent** : none

On exit : X=0 if end of file has not been reached
X<>0 if end of file has been reached
Y is preserved

OSBYTE 128 (&80) Read ADC channel or get buffer status

This is a multi-purpose call with the action specified by the value of the X register.

Entry parameters:	**On exit:**
X=0	Y contains the channel number (1-4) of the channel number last used for an ADC conversion. A value of zero indicates that no conversion has been completed. Bits 0 and 1 of X indicate the status of the two 'fire' buttons.
X=<1–4>	X and Y contain the 16 bit value (X=LSB, Y=MSB) read from the channel specified in X.

X<0 Y=255	X gives the status of various buffers:
X=255	X contains the number of bytes in the keyboard buffer
X=254	X contains the number of bytes in the RS423 input buffer
X=253	X contains the number of bytes free in the RS423 output buffer
X=252	X contains the number of bytes free in the printer buffer
X=251	X contains the number of bytes free in sound channel 0
X=250	X contains the number of bytes free in sound channel 1
X=249	X contains the number of bytes free in sound channel 2
X=248	X contains the number of bytes free in sound channel 3

OSBYTE 129 (&81) Read key with time limit

This call may be used to read a key from the keyboard subject to a specified time limit or to perform a keyboard scan for a specified key depression.

To read a key within a specified time limit:

Entry parameters : X and Y contain the time limit in hundredths of a second.

If a time limit of n hundredths of a second is required, then

X = n AND &FF
Y = n DIV &100

***FX equivalent** : none

On exit : If a character is detected within the time limit:

X=<ASCII value of key detected>
Y=0

If no character is detected within the time limit:

Y=&FF (or &1B if ESCAPE was pressed)

To perform a keyboard scan for a particular character:

Entry parameters : X=<INKEY value of required character> (see table)
Y=&FF

***FX equivalent** : none

On exit : X and Y contain &FF if the key being scanned is depressed.

key	INKEY number		key	INKEY number	
f0	&DF	−33	4	&ED	−19
f1	&8E	−114	5	&EC	−20
f2	&8D	−115	6	&CB	−53
f3	&8C	−116	7	&DB	−37
f4	&EB	−21	8	&EA	−22
f5	&8B	−117	9	&D9	−39
f6	&8A	−118	,	&99	−103
f7	&E9	−23	-	&E8	−24
f8	&89	−119	.	&98	−104
f9	&88	−120	/	&97	−105
A	&BE	−66	[	&C7	−57
B	&9B	−101	\	&87	−121
C	&AD	−83	]	&A7	−89
D	&CD	−51	^	&E7	−25
E	&DD	−35	_	&D7	−41
F	&BC	−68	:	&B7	−73
G	&AC	−84	;	&A8	−88
H	&AB	−85	@	&B8	−72
I	&DA	−38	ESCAPE	&8F	−113
J	&BA	−70	TAB	&9F	−97
K	&B9	−71	CAPS LOCK	&BF	−65
L	&A9	−87	SHIFT LOCK	&AF	−81
M	&9A	−102	CTRL	&FE	−2
N	&AA	−86	SHIFT	&FF	−1
O	&C9	−55	SPACE	&9D	−99
P	&C8	−56	DELETE	&A6	−90
Q	&EF	−17	RETURN	&B6	−74
R	&CC	−52	COPY	&96	−106
S	&AE	−82	↑	&C6	−58
T	&DC	−36	←	&E6	−26
U	&CA	−54	→	&86	−122
V	&9C	−100	↓	&D6	−42
W	&DE	−34	*keypad* 0	&95	−107
X	&BD	−67	*keypad* 1	&94	−108
Y	&BB	−69	*keypad* 2	&83	−125
Z	&9E	−98	*keypad* 3	&93	−109
0	&D8	−40	*keypad* 4	&85	−123
1	&CF	−49	*keypad* 5	&84	−124
2	&CE	−50	*keypad* 6	&E5	−27
3	&EE	−18	*keypad* 7	&E4	−28

key	INKEY number		key	INKEY number	
keypad 8	&D5	−43	*keypad* #	&A5	−91
keypad 9	&D4	−44	*keypad* *	&A4	−92
keypad +	&C5	−59	*keypad* ,	&A3	−93
keypad −	&C4	−60	*keypad* .	&B3	−77
keypad /	&B5	−75	*keypad* RETURN	&C3	−61
			keypad DELETE	&B4	−76

OSBYTE 130 (&82) Read machine high order address

This call provides a 16-bit high order address for filing system addresses which require 32 bits.

Entry parameters : none

***FX equivalent** : none

On exit : X and Y contain the high order address (X=MSB, Y=LSB)

OSBYTE 131 (&83) Read Operating System High Water Mark (OSHWM)

This call is used to return the OSHWM address, which is normally &E00 but may be changed if, for example, the 'hidden RAM' is insufficient for the needs of all active filing systems and languages.

OSBYTE 131 is used by BASIC to initialise the value of pseudo-variable PAGE.

Entry parameters : none

***FX equivalent** : none

On exit : X and Y contain the OSHWM address (X=LSB, Y=MSB)

OSBYTE 132 (&84) Read top of user RAM

This call returns the lowest address used by the VDU driver to store the screen display.

Entry parameters : none

***FX equivalent** : none

On exit : X and Y contain the address (X=LSB, Y=MSB)

Note

If the VDU driver is writing to shadow memory, the address returned in X and Y will be &8000.

OSBYTE 133 (&85) Read top of user RAM for given mode

Whereas OSBYTE 132/&84 returns the top of user RAM for the current mode, OSBYTE 133 returns the top of user RAM for a specified mode change.

Entry parameters : X=<mode number>

***FX equivalent** : none

On exit : X and Y contain the address (X=LSB, Y=MSB)

Note

If the specified mode change would cause the VDU driver to write to shadow memory, the address returned in X and Y will be &8000.

OSBYTE 134 (&86) Read text cursor position

Under normal circumstances, this call returns the current text cursor position. However, if cursor editing is in progress, the position returned is that of the input cursor.

Entry parameters : none

***FX equivalent** : none

On exit : X=<horizontal position>
Y=<vertical position>

Notes

1. This call should not be used inside an interrupt service routine.
2. If there is a scroll 'pending', ie the scroll protect option is enabled and active, the horizontal position returned will be one greater than the displayed position.
3. See OSBYTE 165/&A5 for reading the position of the output cursor in cursor editing mode.

OSBYTE 135 (&87) Read screen mode and character at text cursor position

This call returns the screen mode and, under normal circumstances, the ASCII code of the character at the text cursor position. However, if cursor editing is in progress, the character code returned is that of the character at the input cursor position.

Entry parameters : none

***FX equivalent** : none

On exit : X=<ASCII code> (or zero if unknown)
Y=<screen mode> (ie 0–7 regardless of usage of shadow memory)

OSBYTE 136 (&88) Execute user code

Equivalent to `*CODE`.

Entry parameters : X=<parameter to be passed to user routine>
Y=<parameter to be passed to user routine>

***FX equivalent** : `*FX136,`<param 1>`,`<param 2>

Note

The address of the routine must first be stored in the user vector (USERV, &200–&201).

OSBYTE 137 (&89) Switch cassette motor relay

Equivalent to `*MOTOR`.

Entry parameters : X=0 switches relay OFF
X=1 switches relay ON
Y=0

***FX equivalent** : `*FX137,`<0–1>

On exit : X and Y are undefined

OSBYTE 138 (&8A) Insert character code into buffer

This call inserts a specified value into a specified buffer.

Entry parameters : X=<buffer number> (see OSBYTE 21/&15)
Y=<value>

***FX equivalent** : `*FX138,`<buffer number>`,`<value>

On exit : X and Y are preserved

OSBYTE 139 (&8B) Write Filing System options

Equivalent to `*OPT`.

Entry parameters : X=<option number>
Y=<option value>

***FX equivalent** : `*FX139,`<option number>`,`<option value>

On exit : X and Y are preserved

OSBYTE 140 (&8C) Select Cassette Filing System

Equivalent to `*TAPE`.

Entry parameters : X=0 selects the default baud rate (1200)
X=3 selects 300 baud
X=12 selects 1200 baud

***FX equivalent** : `*FX140,`<0/3/12>

On exit : X and Y are preserved

OSBYTE 141 (&8D) Select ROM Filing System

Equivalent to `*ROM`.

Entry parameters : none

***FX equivalent** : `*FX141`

On exit : X and Y are undefined

OSBYTE 142 (&8E) Enter language ROM

This call enters a language ROM referenced by its ROM socket number.

Entry parameters : X=<socket number>

***FX equivalent** : `*FX142,`<socket number>

Notes

1. This call provides a means of entering language ROMs whose names clash with other ROM names or MOS commands (eg a language called `*FX0`).
2. ROMs which have been 'unplugged' cannot be selected by this call.
3. If a co-processor is available, the designated language will be copied across.

OSBYTE 143 (&8F) Issue paged ROM service request

The MOS requests paged ROMs to perform various functions such as claiming workspace, executing unrecognised MOS commands or responding to BRK instructions. These functions are referred to as services and are requested by paging each ROM in turn and calling its 'service entry point' with a specified 'service type'.

Calls of this nature are mainly of interest to authors of paged ROM software and further information is provided in section F.7.

Entry parameters : X=<service type>
: Y=<argument for service>

***FX equivalent** : `*FX143,`<service type>`,`<argument for service>

On exit : Y may contain a return argument if appropriate (see section F.7).

OSBYTE 144 (&90) Set vertical screen shift and interlace option

Equivalent to `*TV`.

Entry parameters : X=<vertical screen shift>
Y=0 switches the interlace option ON
Y=1 switches the interlace option OFF

***FX equivalent** : `*FX144,`<vertical screen shift>`,`<interlace switch>

On exit : X and Y contain the previous settings.

OSBYTE 145 (&91) Get character from buffer

This call extracts the next character from a specified buffer.

Entry parameters : X=<buffer number> (see OSBYTE 21/&15)

***FX equivalent** : none

On exit : Y contains the extracted character
The carry flag is set if the specified buffer was empty

OSBYTE 146 (&92) – 151 (&97) Access memory mapped I/O areas

Calls in this range are used to read from or write to locations in the memory mapped I/O area (&FC00–&FEFF). Information on the content and significance of locations in this area are given in section F.6.

Read operations:

OSBYTE 146 (&92)	Read from FRED	(&FC00 – &FCFF)
OSBYTE 148 (&94)	Read from JIM	(&FD00 – &FDFF)
OSBYTE 150 (&96)	Read from SHEILA	(&FE00 – &FEFF)

Entry parameters : X=<offset within page>

***FX equivalent** : none

On exit : contains the value that has been read

Write operations:

OSBYTE 147 (&93)	Write to FRED	(&FC00 – &FCFF)
OSBYTE 149 (&95)	Write to JIM	(&FD00 – &FDFF)
OSBYTE 151 (&97)	Write to SHEILA	(&FE00 – &FEFF)

Entry parameters : X=<offset within page>
Y=<value to be written>

***FX equivalent** : `*FX`<147/149/151>`,`<offset within page>`,`<value to be written>

OSBYTE 152 (&98) Examine buffer status

This call returns the status of a specified buffer and, if available, the next character.

Entry parameters : X=<buffer number> (see OSBYTE 21/&15)

***FX equivalent** : none

On exit : The carry flag is set if the buffer is empty otherwise Y contains the next available character

Notes

1. Interrupts must be disabled before using this call and should normally be returned to their previous state immediately afterwards.
2. OSBYTE 152/&98 does not remove the character from the buffer.

OSBYTE 153 (&99) Insert character code into buffer checking for escape

Whereas OSBYTE 138/&8A enables characters to be inserted into any specified buffer, this call relates only to the two input buffers and handles escape characters as if they had been typed or received via the RS423 port.

Entry parameters : X=0 specifies the keyboard buffer
X=1 specifies the RS423 input buffer
Y=<character code to be inserted>

***FX equivalent** : `*FX153,`<buffer number>`,`<character code>

On exit : The carry flag is set if the code could not be inserted into the buffer (eg if the buffer was full).

OSBYTE 154 (&9A) Write video ULA control register

This call writes to register 0 of the video ULA and also to the RAM copy (see OSBYTE 184/&B8) – see section F for further information.

Entry parameters : X=<value to be written>
Y=0

***FX equivalent** : `*FX154,`<value>

On exit : X and Y are preserved

Note

This call also resets the flash counter (see OSBYTE 193/&C1).

OSBYTE 155 (&9B) Write to video ULA palette register and copy

This call writes to register 1 of the video ULA and also to the RAM copy (see OSBYTE 185/&B9) – see section F for further information.

Entry parameters : X=<value to be written>
Y=0

***FX equivalent** : `*FX155,`<value>

On exit : X and Y are preserved

Note

The actual value written is X EOR 7.

OSBYTE 156 (&9C) Read/write serial ACIA control register and copy

This call is used if it is necessary to program the ACIA directly, such as when implementing non-standard RS423 formats.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new register content>
Y=0

The bit settings in the new value are as defined below:

bit1	bit0	baud rate
0	0	clock / 1 = baud rate × 64
0	1	clock / 16 = baud rate × 4
1	0	clock / 64 = baud rate × 1
1	1	reset transmit, receive and control registers

bit4	bit3	bit2	word length	parity	stop bits
0	0	0	7	even	2
0	0	1	7	odd	2
0	1	0	7	even	1
0	1	1	7	odd	1
1	0	0	8	none	2
1	0	1	8	none	1
1	1	0	8	even	1
1	1	1	8	odd	1

bit6	bit5	transmission control
0	0	RTS low, transmit interrupt disabled
0	1	RTS low, transmit interrupt enabled
1	0	RTS high, trasmit interrupt disabled
1	1	RTS at break level, transmit interrupt disabled

bit7	receive interrupt
0	disabled
1	enabled

On exit : X=<old register contents>
Y is preserved

OSBYTE 157 (&9D) Write byte across Tube

This call is a faster alternative to the use of OSBPUT (see section G) when using a co-processor.

Entry parameters : X=<byte to be written>
Y=<file handle>

***FX equivalent** : `*FX157,`<byte>`,`<file handle>

On exit : X and Y are undefined

OSBYTE 158 (&9E) and 159 (&9F) reserved for the speech system.

OSBYTE 160 (&A0) Read VDU variable value

The VDU driver uses a number of locations in RAM to store transient information such as the display mode. Further information is given in section E.4.

Entry parameters : X=<VDU variable number>

***FX equivalent** : none

On exit : X and Y contain the value of the variable (X=LSB, Y=MSB)

OSBYTE 161 (&A1) Read CMOS RAM

This call provides read access to any of the locations in the CMOS RAM.

Entry parameters : X=<CMOS RAM location>

***FX equivalent** : none

On exit : Y=<contents of CMOS RAM location>

OSBYTE 162 (&A2) Write CMOS RAM

This call provides write access to any of the locations in the CMOS RAM with the exception of location 0, which is protected.

Entry parameters : X=<CMOS RAM location>
Y=<value>

***FX equivalent** : `*FX162,`<CMOS RAM location>`,`<value>

On exit : X is preserved
Y is undefined

OSBYTE 163 (&A3) is reserved for applications software.

OSBYTE 164 (&A4) Check processor type

This call is used by filing systems to determine if a piece of code to be `*RUN` is suitable for execution in an I/O processor.

Entry parameters : X=<LSB of code address>
Y=<MSB of code address>

***FX equivalent** : none

On exit : X and Y are undefined

OSBYTE 165 (&A5) Read output cursor position

OSBYTE 165 may be used to determine the position of the output cursor while cursor editing is in progress.

Entry parameters : none

***FX equivalent** : none

On exit : X=<horizontal position>
Y=<vertical position>

OSBYTE 166 (&A6) and 167 (&A7) Read start address of MOS variables

These calls access the address of the first memory location used by the MOS to store its internal variables. This address is written only on soft reset.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : *For OSBYTE 166:*
X=<LSB of start address>
Y=<MSB of start address>

For OSBYTE 167:
X=<MSB of start address>
Y=<contents of next location>

OSBYTE 168 (&A8) and 169 (&A9) Read address of ROM pointer table

These calls access the start address of the table of extended vectors. Each extended vector is three bytes in length (two for the location and one for the corresponding ROM number).

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : *For OSBYTE 168:*
X=<LSB of pointer table address>
Y=<MSB of pointer table address>

For OSBYTE 169:
X=<MSB of pointer table address>
Y=<contents of next location>

OSBYTE 170 (&AA) and 171 (&AB) Read address of ROM information table

The ROM information table is a group of 16 bytes – one for each of the 16 paged ROMs. Each byte contains 0 if the ROM socket is empty, unplugged or not in the correct format for a paged ROM, or the corresponding ROM type byte (location &8006 from the paged ROM). These calls access the start address of the table.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : *For OSBYTE 170:*
X=<LSB of information table address>
Y=<MSB of information table address>

For OSBYTE 171:
X=<MSB of information table address>
Y=<contents of next location>

OSBYTE 172 (&AC) and 173 (&AD) Read address of keyboard translation table

The keyboard translation table contains the ASCII values corresponding to the various internal key numbers. These calls access the start address of the table.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : *For OSBYTE 172:*
X=<LSB of table address>
Y=<MSB of table address>

For OSBYTE 173:
X=<MSB of table address>
Y = <contents of next location>

OSBYTE 174 (&AE) and 175 (&AF) Read address of VDU variables origin

Internal VDU variables are held in a table whose start address is accessed by these calls.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : *For OSBYTE 174:*
X=<LSB of origin address>
Y =<MSB of origin address>

For OSBYTE 175:
X=<MSB of origin address>
Y=<contents of next location>

OSBYTE 176 (&B0) Read/Write CFS timeout counter

The CFS timeout counter is decremented once for each vertical sync pulse and is used to time interblock gaps and leader tones. It may be read or set explicitly using this call.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX176,`<value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 177 (&B1) Read/write input source

Input may be taken from either the keyboard or the RS423 port. This call provides a means of determining the current source or switching between the two.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 for keyboard input
X=1 for RS423 input
Y=0

***FX equivalent** : `*FX177,`<0/1>

On exit : X=<old input source>
Y=<contents of next location>

OSBYTE 178 (&B2) Read/write keyboard semaphore

This call provides a means of enabling or disabling keyboard interrupts.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 ignores keyboard interrupts
X=255 enables keyboard interrupts
Y=0

***FX equivalent** : `*FX178,`<0/255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 179 (&B3) Read/write ROM polling semaphore

This call enables the ROM polling semaphore to be set explicitly (see OSBYTE calls 22/&16 and 23/&17).

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX179,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 180 (&B4) Read/write Operating System High Water Mark (OSHWM)

This call provides a means of reading OSHWM (equivalent to OSBYTE 131/&83) and resetting it to a different value.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX180,`<new value>

On exit : X=<MSB of OSHWM address> (LSB is always 0)
Y=<contents of next location>

OSBYTE 181 (&B5) Read/write RS243 input interpretation status

By default, RS423 input ignores the current escape character, does not expand soft key codes and does not generate events. This call provides a means of reading the current status (ie default or otherwise) and setting it such that RS423 input behaves like keyboard input.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 simulates keyboard input
X=1 sets default status
Y=0

***FX equivalent** : `*FX181,<0/1>`

On exit : X=<old interpretation status>
Y=<contents of next location>

OSBYTE 182 (&B6) Read NOIGNORE state

This call provides a means of determining the presence (or absence) of a printer ignore character.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X<0 (ie top bit set) indicates that there is no current printer ignore character
X>=0 indicates the presence of a printer ignore character (which can be determined using OSBYTE 246/&F6 (see below)

OSBYTE 183 (&B7) Read/write cassette/ROM filing system switch

This call provides a means of determining (and changing) the selection of either the cassette or ROM Filing System.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 specifies cassette
X=2 specifies ROM
Y=0

***FX equivalent** : `*FX183,<0/2>`

On exit : X=<old setting>
Y=<contents of next location>

OSBYTE 184 (&B8) Read OS copy of video ULA control register

This call is the read equivalent of OSBYTE 154/&9A although the location read is the operating system's internal copy of the register.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<last value written to ULA control register>
Y=<contents of next location>

Note

This call must not be used for writing purposes as to do so would make the OS copy of the register inconsistent with the actual register.

OSBYTE 185 (&B9) Read OS copy of video ULA palette register

This call is the read equivalent of OSBYTE 155/&9B although the location read is the operating system's internal copy of the register.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<last value written to video ULA palette register>
Y=<contents of next location>

Note

This call must not be used for writing purposes as to do so would make the OS copy of the register inconsistent with the actual register.

OSBYTE 186 (&BA) Read ROM number active at last BRK

This call returns the ROM socket number of the language or filing system active during execution of a BRK instruction.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<ROM number>
Y=<contents of next location>

OSBYTE 187 (&BB) Read ROM number of socket containing BASIC

This call is provided for the purpose of identifying the BASIC ROM, which is the only language ROM without a service entry point.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<ROM socket number> (255=no BASIC ROM)
Y=<contents of next location>

OSBYTE 188 (&BC) Read current ADC channel number

This call returns the number of the ADC channel currently being sampled.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<channel number>
Y=<contents of next location>

Note

This call should not be used to write the current ADC channel – use OSBYTE 17/&11.

OSBYTE 189 (&BD) Read maximum ADC channel number

This call returns the current maximum ADC channel number, as set by OSBYTE 10/&0A.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<maximum channel number>
Y=<contents of next location>

Note

This call should not be used to write the maximum channel number – use OSBYTE 16/&10.

OSBYTE 190 (&BE) Read/write ADC conversion type

This call provides a means of specifying the resolution to be used in ADC conversions.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 sets the default resolution (12 bits)
X=8 sets 8 bit resolution
X=12 sets 12 bit resolution explicitly
Y=0

***FX equivalent** : `*FX190,`<0/8/12>

On exit : X=<old resolution>
Y=<contents of next location>

OSBYTE 191 (&BF) Read/write RS423 busy flag

This call is used to either read or reset the RS423 busy flag which is used to ensure that the serial interface hardware is not taken over by the CFS while the RS423 driver is using it (or vice versa).

Entry parameters : *To read:* X=0
Y=255

To write: X=0 to set 'busy'
X=128 to set 'free'
Y=127

***FX equivalent** : `*FX191,0,127` to set 'busy'
`*FX191,128,127` to set 'free'

On exit : X<0 (ie bit 7 set) indicates RS432 'free'
X>=0 (ie bit 7 clear) indicates RS423 'busy'
Y=<contents of next location>

OSBYTE 192 (&C0) Read serial ACIA control register

This call is the read equivalent of OSBYTE 156/&9C.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<old register content>
Y=<contents of next location>

Note

This call should not be used to write to the serial ACIA control register as to do so would make the OS copy of the register inconsistent with the actual register.

OSBYTE 193 (&C1) Read/write flash counter

The flash counter contains the number of vsync periods which must elapse before the next change of flashing colour. This call is used either to determine the current value or to reset it to a new value.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX193,`<new value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 194 (&C2) Read/write duration of first colour

This call provides a means of reading or resetting the number of fiftieths of a second for which the first of a pair of flashing colours will be displayed.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 sets an infinite duration
X=<1–255> sets a duration of n fiftieths of a second
Y=0

***FX equivalent** : `*FX194,`<new value>

On exit : X=<old value>
Y=<contents of next location>

Note

For writing, this call is equivalent to OSBYTE 9.

OSBYTE 195 (&C3) Read/write duration of second colour

This call provides a means of reading or resetting the number of fiftieths of a second for which the second of a pair of flashing colours will be displayed.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 sets an infinite duration
X=<1–255> sets a duration of n fiftieths of a second
Y=0

***FX equivalent** : `*FX195,`<new value>

On exit : X=<old value>
Y=<contents of next location>

Note

For writing, this call is equivalent to OSBYTE 10/&0A.

OSBYTE 196 (&C4) Read/write keyboard auto-repeat delay

This call provides a means of reading or resetting the delay which must elapse before a key will start to auto-repeat.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 disables auto-repeat
X=<1–255> sets the delay to n hundredths of a second
Y=0

***FX equivalent** : `*FX196,`<new value>

On exit : X=<old value>
Y=<contents of next location>

Note

This call is used by OSBYTE 11/&0B.

OSBYTE 197 (&C5) Read/write keyboard auto-repeat rate

This call provides a means of reading or resetting the rate at which keys auto-repeat.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 resets the keyboard auto-repeat delay and rate to their default values
X=<1–255> sets the repeat rate to n hundredths of a second
Y=0

***FX equivalent** : `*FX197,`<new value>

On exit : X=<old value>
Y=<contents of next location>

Note

This call is used by OSBYTE 12/&0C.

OSBYTE 198 (&C6) Read/write *EXEC file handle

This call provides a means of reading or changing the handle currently associated with a `*EXEC` file.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new handle>
Y=0

***FX equivalent** : `*FX198,`<new handle>

On exit : X=<old handle>
Y=<contents of next location>

Note

If the file handle is being changed, the corresponding file must be open or channel errors will result.

OSBYTE 199 (&C7) Read/write *SPOOL file handle

This call provides a means of reading or changing the handle currently associated with a `*SPOOL` file.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new handle>
Y=0

***FX equivalent** : `*FX199,`<new handle>

On exit : X=<old handle>
Y=<contents of next location>

Note

If the file handle is being changed, the corresponding file must be open or channel errors will result.

OSBYTE 200 (&C8) Read/write BREAK and ESCAPE effect

This call provides a means of reading or changing the effect of BREAK and ESCAPE.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value> with bits set as follows:

bit0	effect
0	normal ESCAPE action
1	ESCAPE disabled unless caused by OSBYTE 125/&7D

bit1	effect
0	normal BREAK action
1	memory cleared on BREAK

Y=0

***FX equivalent** : `*FX200,`<value>

On exit : X=<old value>
Y= <contents of next location>

OSBYTE 201 (&C9) Read/write keyboard status

This call provides a means of reading and changing the keyboard status (ie whether it is enabled or disabled).

Entry parameters : *To read:* X=0
Y=255

To write: X=0 enables keyboard input
X=<1–255> effectively disconnects the keyboard by preventing the keyboard interrupt service routine from placing characters in the keyboard buffer
Y=0

***FX equivalent** : `*FX201,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

Note

This call is used by the Econet `*REMOTE` facility.

OSBYTE 202 (&CA) Read/write keyboard status byte

The keyboard status byte holds information on the current status of the keyboard, such as the settings of caps lock and shift lock. This call provides a means of reading and changing these settings from software.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value> (see bit settings on exit)
Y=0

***FX equivalent** : `*FX202,`<value>

On exit : X=<old value>, the bits settings having the following significance:

bit	indication when set
3	SHIFT depressed
4	CAPS LOCK disengaged
5	SHIFT LOCK disengaged
6	CTRL depressed
7	SHIFT enabled (see note)

Y=<contents of next location>

Note

Bit 7 is used to indicate the use of SHIFT+CAPS LOCK (ie as for `*CONFIGURE SHCAPS`) or SHIFT+SHIFT LOCK (in which case normal key depressions produce SHIFTed characters and key depressions in conjunction with SHIFT produce normal characters). This latter facility is implemented by means of `*FX202,144` (or the equivalent OSBTYE call) and cannot be set from the keyboard.

OSBYTE 203 (&CB) Read/write RS423 input buffer minimum space

By default, the RS423 input routine attempts to halt input when there are nine free spaces left in the input buffer. This call allows the value to be read or changed if neccessary.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX203,`<value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 204 (&CC) Read/write RS423 ignore flag

This call is used to read or change the flag which indicates whether RS423 input is to be ignored.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 enables RS423 input
X=<1–255> ignores subsequent RS423 input
Y=0

***FX equivalent** : `*FX204,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

Note

1. Although this call stops data being placed in the RS423 input buffer, data is still received normally and errors will still generate events (if enabled).
2. This command should always be used after routing RS423 output to the cassette port (see OSBYTE 205/&CD). This prevents written data being read back into the RS423 input buffer.

OSBYTE 205 (&CD) Read/write RS423 destination

RS423 output may be routed to either the cassette or serial interface hardware and this call provides a means of determining the current destination or changing it.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 sets destination to RS423
X=64 sets destination to cassette
Y=0

***FX equivalent** : `*FX205,`<0/64>

On exit : X=<old value>
Y=<contents of next location>

Note

Any change made by this call will come into effect only after use of OSBYTE 7/8.

OSBYTE 206 (&CE) Read/write Econet OS call interception status

By default, OSBYTE and OSWORD calls are handled by the appropriate MOS routines but, using this call, it is possible to have recognised calls indirected via the Econet vector (&224,&225) if required.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 selects normal MOS routines
X=128 selects indirection via the Econet vector
Y=127

***FX equivalent** : `*FX206,`<0/128>`,127`

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 207 (&CF) Read/write Econet input interpretation status

By default, the MOS character input routine (OSRDCH) takes values from the stream specified by OSBYTE 2 but, using this call, it is possible to take input from the routine pointed to by the Econet vector.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 sets normal input
X=128 sets input via the Econet vector
Y=127

***FX equivalent** : `*FX207,`<0/128>`,127`

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 208 (&D0) Read write Econet output interpretation status

By default, character output via OSWRCH is normally sent to the stream(s) specified by OSBYTE 3 but, using this call, it is possible to have output directed to the routine pointed to by the Econet vector.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 sets normal output
X=128 sets output via the Econet vector
Y=127

***FX equivalent** : `*FX208,`<0/128>`,127`

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 209 (&D1) is reserved for the speech system.

OSBYTE 210 (&D2) Read/write sound suppression status

Sound output is enabled by default and this call provides a means of reading or changing the sound suppression byte.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 enables sound output
X=<1–255> disables sound output
Y=0

***FX equivalent** : `*FX210,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 211 (&D3) Read/write BELL channel

The BELL (CTRL+G) sound is output on channel 3 by default. This call provides a means of determining the current channel or changing it if required.

Entry parameters : *To read:* X=0
Y=255

To write: X=<channel number>
Y=0

***FX equivalent** : `*FX211,`<0–3>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 212 (&D4) Read/write BELL sound information

`*CONFIGURE LOUD` and `*CONFIGURE QUIET` set default values for the output of the BELL sound (216 and 240 respectively). This command provides a means of reading the current setting or changing it if required.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value> as defined below:

To specify a fixed amplitude (in the range −15 to 0) the new value should be set to:

256+((amplitude−1)*8)

To specify an envelope (in the range 1–16) the new value should be set to:

(envelope number−1)*8

Y=0

***FX equivalent** : `*FX212,`<value>

On exit : X=<old value>
Y=<contents of next location>

Note

The factor of 8 in each of the above calculations is to ensure that the least significant three bits of the sound information byte are unaffected by the specification of either an amplitude or an envelope. If required, these bits may be used to hold the H and S components of the expanded SOUND command associated with BELL.

OSBYTE 213 (&D5) Read/write BELL frequency

This command provides a means of reading or changing the frequency associated with the BELL sound.

Entry parameters : *To read:* X=0
Y=255

To write: X=<frequency> (0–255)
Y=0

***FX equivalent** : `*FX213,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 214 (&D6) Read/write BELL duration

This call provides a means of reading or changing the duration of the sound associated with BELL.

Entry parameters : *To read:* X=0
Y=255

To write: X=<duration>
Y=0

***FX equivalent** : `*FX214,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 215 (&D7) Read/write startup message suppression status

By default, the MOS displays a startup message when the system is reset. This call provides a means of reading this setting and changing it if necessary.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value> with bits set as below:

bit7	effect
0	suppresses the startup message
1	displays startup message as normal

bit0	effect
0	causes the computer to lock up if there is an error in a ROM !BOOT file
1	causes the computer to lock up if there is an error in a disc !BOOT file

Y=1	to change !BOOT error action only
Y=128	to change startup message suppression only
Y=129	to change both settings

***FX equivalent** : `*FX215,`<new value>`,`<mask>

On exit : X=<old value>
Y = <contents of next location>

OSBYTE 216 (&D8) Read/write length of soft key string

When a soft key is being read by OSRDCH, the MOS keeps a count of the number of characters left to be read. This call provides a means of reading the count and changing it if necessary. The latter action is not recommended.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value for counter>
Y=0

***FX equivalent** : `*FX216,`<value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 217 (&D9) Read/write paged mode line count

In the paged output mode, the display is prevented from scrolling (awaiting the depression of SHIFT) when approximately 75% of the maximum number of lines for the current mode (or text window) have been output. The number of lines printed since the last page halt is maintained in the location accessed by this call and it may be either read or changed (normally to 0 before requesting user input).

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value for count>
Y=0

***FX equivalent** : `*FX217,`<value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 218 (&DA) Read/write bytes in VDU queue

The location accessed by this call is a 2s complement count of the number of characters remaining to be passed to the VDU driver in order to complete the current command. The call may be used to read the value or to change it (normally to zero, which has the effect of abandoning an incomplete VDU command – see section E).

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX218,`<value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 219 (&DB) Read/write TAB key code

By default, the TAB key generates ASCII 9. This call provides a means of reading the ASCII code currently assigned to TAB and of changing it to another value

Entry parameters : *To read:* X=0
Y=255

To write: X=<new code>
Y=0

***FX equivalent** : `*FX219,`<0–255>

On exit : X=<old code>
Y=<contents of next location>

Note

If the code assigned to TAB is in the range 128–143, it will function as the soft key numbered (code−128).

OSBYTE 220 (&DC) Read/write ESCAPE character

ESCAPE (ASCII 27) is the default escape character. This call provides a means of reading the ASCII code of the current escape character and of redefining it it required.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new code>
Y=0

***FX equivalent** : `*FX220,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 221 (&DD) Read/write interpretation of input values 192-207

OSBYTE 222 (&DE) Read/write interpretation of input values 208-223

OSBYTE 223 (&DF) Read/write interpretation of input values 224-239

OSBYTE 224 (&E0) Read/write interpretation of input values 240-255

These calls provide a means of reading and changing the way in which input values in the range 192–255 are interpreted.

These codes cannot be generated from the main keyboard but may be produced if the numeric keypad is re-based (see OSBTYE 238/&EE) or if the values are inserted into the input buffer directly (see OSBYTE calls 138/&8A and 153/&99).

Entry parameters : *To read:* X=0
Y=255

To write: X=<param> as below:

<param>	**effect**
0	causes the code to be ignored
1	causes the code to generate the string assigned to soft key (code MOD 16)
2–255	causes the code to generate the value <param> + (code MOD 16)

Y=0

***FX equivalent** : *FX<221/222/223/224>,<param>

On exit : X=<old value>
Y=<contents of next location>

Note

Soft key numbers 11 – 15 correspond to [COPY], [←], [→], [↓] and [↑] respectively provided that they have been enabled using OSBYTE 4.

OSBYTE 225 (&E1) Read/write soft key interpretation
OSBYTE 226 (&E2) Read/write SHIFT+soft key interpretation
OSBYTE 227 (&E3) Read/write CTRL+soft key interpretation
OSBYTE 228 (&E4) Read/write SHIFT+CTRL+soft key interpretation

These calls provide a means of reading and changing the effect of function key depressions both on their own and in conjunction with SHIFT and/or CTRL.

Entry parameters : *To read:* X=0
Y=255

To write: X=<param> as below:

<param>	**effect**
0	causes the key depression to be ignored
1	causes the key depression to generate the string assigned to soft key (code MOD 16)
2–255	causes the key depression to generate the value <param> + (code MOD 16)

Y=0

***FX equivalent** : `*FX<225/226/227/228>,`<param>

On exit : X=<old value>
Y=<contents of next location>

Notes

1. Soft key numbers 11 – 15 correspond to COPY, ←, →, ↓ and ↑ respectively provided that they have been enabled using OSBYTE 4.
2. These calls also affect the interpretation of codes in the range 128–191 in a similar manner to OSBYTE calls 221–224 (&DD–&E0).

OSBYTE 229 (&E5) Read/write ESCAPE key status

This call provides a means of enabling or disabling the generation of escape conditions (caused by the depression of the current escape character or by the occurrence of the corresponding ASCII code in the input buffer).

Entry parameters	:	*To read:*	X=0 Y=255	
		To write:	X=0	enables escape condition generation
			X=<1–255>	disables escape condition generation and causes the current escape character to generate its ASCII code
			Y=0	
***FX equivalent**	:	`*FX229,`<0–255>		
On exit	:	X=<old value> Y=<contents of next location>		

Note

See also OSBYTE 200/&C8.

OSBYTE 230 (&E6) Read/write ESCAPE effects

By default, the acknowledgement of an escape condition produces the following effects:

– all active buffers are flushed;
– any currently open `*EXEC` file is closed;
– the VDU queue is cleared;
– the VDU line count used in paged mode is cleared;
– any current soft key expansion is cancelled;
– any sound being produced is terminated.

This call provides a means of determining whether escape effects are currently enabled or disabled and of changing the setting if required.

Entry parameters	:	*To read:*	X=0 Y=255	
		To write:	X=0	enables the side effects
			X=<1–255>	disables the side effects
			Y=0	

***FX equivalent** : `*FX230,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 231 (&E7) Read/write IRQ bit mask for user 6522

OSBYTE 232 (&E8) Read/write IRQ bit mask for 6850

OSBYTE 233 (&E9) Read write IRQ bit mask for system 6522

These calls provide a means of enabling or disabling MOS handing of interrupts associated with the user port (user 6522), RS423 interface (6580) and other aspects of the computer's internal operation (system 6522). Further details can be found in the Advanced Reference Manual.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX`<231/232/233>`,`<new value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 234 (&EA) Read flag indicating Tube presence

This call provides a means of determining whether a Tube system (ie an internal or external co-processor) is present in the machine.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=0 indicates no Tube present
X=255 indicates Tube presence
Y=<contents of next location>

Note

Although this call provides a write function it should not be used.

OSBYTE 235 (&EB) is reserved for the speech system.

OSBYTE 236 (&EC) Read/write character destination status

This call is used by OSBYTE 3.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new destination status> (as for OSBYTE 3)
Y=0

***FX equivalent** : `*FX236,`<new value>

On exit : X=<old destination>
Y=<contents of next location>

OSBYTE 237 (&ED) Read/write cursor editing status

This call is used by OSBYTE 4.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new status> (as for OSBYTE 4)
Y=0

***FX equivalent** : `*FX237,`<new value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 238 (&EE) Read/write numeric keypad interpretation

By default, the keys in the numeric keypad have a 'base number' of 48, ie they generate codes which are displacements from 48 (keypad 0):

−5	−3	−1	−6
+7	+8	+9	−13
+4	+5	+6	+79
+1	+2	+3	−4
+0	−2	−35	

This call provides a means of reading the current base number and changing it if necessary.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new base number>
Y=0

***FX equivalent** : `*FX238,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

Notes

1. Unlike the soft keys, the numeric keypad base number may be set to any value in the range 0–255. (If a generated code lies outside this range it is reduced MOD 256).
2. If a generated code lies in the range 128–143, the corresponding key will behave as a soft key, as described under OSBYTE 219/&DB (above).

OSBYTE 239 (&EF) Read/write *SHADOW state

This call provides a read equivalent to OSBYTE 114/&72.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=0 indicates shadow mode ON (ie shadow memory is used regardless of current mode number)
X<>0 indicates shadow mode OFF (ie use of shadow memory is dependent upon mode number)
Y= <contents of next location>

OSBYTE 240 (&F0) Read country flag

The location accessed by this call contains a coded value representing the country for which the MOS was produced.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X is undefined
Y=0 indicates UK
Y=1 indicates USA

Notes

1. This call provides a write function although its use is limited.
2. Contact Acorn for details of the countries designated by other values.

OSBYTE 241 (&F1) Read/write user flag

This call is used by OSBYTE 1.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new value>
Y=0

***FX equivalent** : `*FX241,`<new value>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 242 (&F2) Read copy of serial processor ULA register

This call returns the content of the RAM copy of the serial ULA register (ie the last value written). It should not be used for writing as this would cause the Operating system's copy of the register to be at variance with register content itself.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<old value>
Y=<contents of next location>

Note

The values held in the serial ULA register can be changed explicitly using the following OSBYTE calls:

– 137 (&89) motor control;
– 205 (&CD) cassette/RS423 select;
– 7 (&07) RS423 receive rate;
– 8 (&08) RS423 transmit rate.

OSBYTE 243 (&F3) Read timer switch state

The operating system maintains two internal clocks which are read and updated alternately. This call is used to determine the clock currently being updated – it should not be used for writing.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=5 indicates the clock value stored in locations &292 – &296
X=10 indicates the clock value stored in locations &297 – &29B
Y= <contents of next location>

Note

These clock values are completely independent of the clock/calendar maintained in the CMOS RAM.

OSBYTE 244 (&F4) Read/write soft key consistency flag

The soft key buffer is said to be 'consistent' at all times other than when it is being updated. The flag accessed by this call is zero if the buffer is consistent and non-zero otherwise. The MOS examines the state of the flag during soft resets and clears the soft key buffer only if it is inconsistent.

This call is normally used for reading although writing a value other than zero will force clearance of the soft key buffer on soft break.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 specifies a consistent soft key buffer
X<>0 specifies an inconsistent soft key buffer
Y=0

***FX equivalent** : `*FX244,`<0–255>

On exit : X=<old setting>
Y=<contents of next location>

OSBYTE 245 (&F5) Read printer driver type

This call provides a means of reading the current printer driver number.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=<printer driver number> (see OSBYTE 5)
Y=<contents of next location>

Note

Do not use this call to select a printer driver – use OSBYTE 5.

OSBYTE 246 (&F6) Read/write printer ignore character

This call is used by OSBYTE 6 and is used to read or change the current printer ignore character.

Entry parameters : *To read:* X=0
Y=255

To write: X=<new ignore character>
Y=0

***FX equivalent** : `*FX246,`<new ignore character>

On exit : X=<old ignore character>
Y=<contents of next location>

OSBYTE 247 (&F7)
OSBYTE 248 (&F8)
OSBYTE 249 (&F9) — Intercept BREAK vector

These three calls (which should be used as a group) provide a means of passing control to a user routine on depression of BREAK.

Entry parameters : *for OSBYTE 247:* X=&4C (a JMP instruction)
Y=0
for OSBYTE 248: X=<LSB of user code address>
Y=0
for OSBYTE 249: X=<MSB of user code address>
Y=0

***FX equivalent** : `*FX247,&4C`
`*FX248,`<LSB>
`*FX249,`<MSB>

On each exit : X=<old value>
Y=<contents of next location>

Note

The user-supplied code will be entered twice for each break:

– with C=1, before the reset message is printed and the Tube is initialised;
– with C=0, after the above.

The user routine should preserve all registers and exit with an RTS.

OSBYTE 250 (&FA) Read memory written by VDU driver

Writing to main or shadow memory by the VDU driver is specified using OSBYTE 112/&70. This call returns the current setting.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=0 indicates writing to memory specified by the current mode
X=1 indicates writing to main memory
X=2 indicates writing to shadow memory
Y=<contents of next location>

OSBYTE 251 (&FB) Read memory displayed

The display of main or shadow memory is specified using OSBYTE 113/&71. This call returns the current setting.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=0 indicates display of memory specified by the current mode number
X=1 indicates display of main memory
X=2 indicates display of shadow memory
Y=<contents of next location>

OSBYTE 252 (&FC) Read/write current language ROM number

This call provides a means of determining and/or changing the current language ROM number (as set by OSBYTE 142/&8E). If the ROM number is changed using this call, the language it contains will be selected on a soft break.

Entry parameters : *To read:* X=0
Y=255

To write: X=<rom number>
Y=0

***FX equivalent** : `*FX252,`<rom number>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 253 (&FD) Read last BREAK type

This call returns the type of the last BREAK performed.

Entry parameters : X=0
Y=255

***FX equivalent** : none

On exit : X=0 indicates a soft break
X=1 indicates a power-on reset
X=2 indicates a hard break
Y=<contents of next location>

OSBYTE 254 (&FE) Set effect of SHIFT on numeric keypad

By default, neither the shift lock nor depression of SHIFT has any effect upon the numeric keypad – this call enables the effect of SHIFT (or shift lock) to be enabled or disabled. If enabled, depression of SHIFT with any of the keys in the keypad will produce the shifted character on the corresponding keys in the main keyboard.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 enables the effect of SHIFT
X=<1–255> disables effect of SHIFT
Y=0

***FX equivalent** : `*FX254,`<0–255>

On exit : X=<old value>
Y=<contents of next location>

OSBYTE 255 (&FF) Read/write startup options

This call provides a means of reading and interchanging the effects of BREAK and SHIFT+BREAK.

Entry parameters : *To read:* X=0
Y=255

To write: X=0 interchanges the effects
X=8 restores normal functions
Y=247

***FX equivalent** : `*FX255,`<0/8>`,247`

On exit : X=<old setting>
Y=<contents of next location>

D.3 Executing other MOS commands (OSWORD)

The OSWORD routine is used to perform a variety of operating system functions which either require or return more than two bytes of information.

Call address &FFF1
Indirected via &20C (WORDV)

As with OSBYTE (described in the previous section) each OSWORD function is identified by a call number in the range 0 – 255:

– call numbers 0 to 15 are performed by the MOS and are described in section D.3.2;
– call numbers 16 to 223 are passed to the paged ROMs;
– call numbers 224 to 255 are passed to the user vector.

The call number is passed to OSWORD in the accumulator, and the X and Y registers are used, not for parameters, but as a pointer to a 'parameter block' in memory:

X=<LSB of parameter block address>
Y=<MSB of parameter block address>

The parameter block is used for passing data to the routine and/or holding returned values; parameter blocks vary in size and content according to the call in question and details are given in the descriptions which follow.

OSWORD calls are normally used in assembly language programs:

```
<instructions to set up parameter block>
.
.
LDA #<call number>
LDX #<LSB of parameter block address>
LDY #<MSB of parameter block address>
JSR &FFF1
```

and it is left to the user to define the parameter block in a safe area of memory.

High level languages such as BASIC implement OSWORD calls by means of statements or functions but OSWORD may be used explicitly if required:

```
DIM pblock% <size>
.
.
<instructions to set up parameter block>
.
.
A%=<call number>
X%=pblock% MOD 256
Y%=pblock% DIV 256
Z%=USR(&FFF1)
```

Note the use of a DIM statement to reserve <size> bytes for the parameter block; use of an explicit address is permitted but not recommended.

There is no equivalent of a `*FX` command to allow OSWORD calls to be made from the command mode of languages.

D.3.1 OSWORD call summary

The sixteen OSWORD calls handled by the MOS are given below in 'functional' groups – a full description of each one is provided in section D.3.2.

Calls marked ☐ are new calls, not implemented on previous BBC microcomputer systems.

Character input

A single OSWORD call provides a general character input routine:

dec.	hex.	description
0	00	Read line from input stream to memory

CMOS clock

The CMOS clock is a calendar/clock chip powered by a trickle-charged battery, which allows it to continue to function when the computer is turned off. The calendar automatically adjusts for leap years on a four-year cycle; it will therefore calculate correctly that the year 2000 is, in fact, a leap-year whilst most century years are not.

Two OSWORD calls are associated with access to the clock maintained in the CMOS RAM:

dec.	hex.	description
14 ☐	0E	Read the CMOS clock
15 ☐	0F	Write the CMOS clock

Interval timer

The interval timer provides an easy way of generating repeated interrupts at a frequency decided by the user. These interrupts generate timer events, which can be enabled or disabled without affecting operation of the interval timer.

The interval timer uses a five byte counter which is incremented every 10 milliseconds. When the counter has reached &FFFFFFFFFF the next increment sets it to zero, generating an interrupt. Thus, the time to the next interrupt is:

(&10000000000 − <variable>) / 100 seconds.

OSWORD calls associated with access to the interval timer are:

dec.	hex.	description
3	03	Read interval timer
4	04	Write interval timer

I/O processor memory access

The OSWORD calls below should be used to guarantee that your programs will read/write memory in the I/O processor even if they are running in a second processor.

dec.	hex.	description
5	05	Read byte from I/O processor memory
6	06	Write byte to I/O processor memory

Screen operations

The OSWORD calls below are associated with screen handling:

dec.	hex.	description
9	09	Read pixel logical colour
10	0A	Read a character definition
11	0B	Read the palette
12	0C	Write the palette
13	0D	Read current and previous graphics cursor positions

Sound generation

The computer can produce sounds through the speaker mounted to the right of the keyboard ruler strip, or through suitable audio equipment connected to the audio output at the rear of the case. These sounds consist of up to three tones (notes) and noise, mixed together to produce chords, piano-like harmonics, explosions etc. Although the basic components of these sounds (tones and noise) are simple, the MOS makes clever use of sound queuing and time-dependent shaping of the sounds' pitch and volume (envelopes) to improve the variety of effects that can be produced with fairly simple commands. The OSWORD calls associated with the generation of sounds are:

dec.	hex.	description
7	07	Produce a sound
8	08	Define a sound envelope

System clock

The system clock is implemented as a pair of 5 byte locations which are incremented 100 times a second (during the system's 10 millisecond interrupt). These locations are set to zero by turning the system off or performing a hard break (CTRL+BREAK). The system clock is completely independent of the clock/calendar maintained by the CMOS RAM. The two OSWORD calls associated with the system clock are:

dec.	hex.	description
1	01	Read system clock
2	02	Write system clock

D.3.2 OSWORD call descriptions

OSWORD calls 0 to 16 are described below. The information presented for each call is as follows:

– *Call number and function*

  The call number is specified in decimal (and hexadecimal); the function is a brief statement of the use of the call.

– *Description*

  a description of the effect of the call.

– *Parameter block size*

  This entry specifies the number of bytes which must be reserved for parameters to the call and/or values returned from the call.

– *Parameter block format*

  This entry describes the way in which the bytes in the parameter block are interpreted on entry to the routine. XY denotes the memory location specified in the X and Y registers; XY+1 denotes the next location etc.

  Only those locations in the parameter block which are significant on entry to the routine are described under this heading – the content of any remaining locations is immaterial.

– *Conditions on exit*

  Where appropriate, this entry describes specific conditions on exit from the routine. XY, XY+1 etc are as defined above and, unless specified otherwise, locations which are not mentioned explicitly will remain unchanged on exit from the routine.

If necessary, any additional information is given under the heading *Notes*.

OSWORD 0 (&00) Read line from input stream to memory

This routine reads characters from the current input stream (as specified by OSBYTE 2) to an area of memory (ie a buffer) specified by the user. Exit from the routine is made when RETURN is pressed or when an escape condition is generated (eg ESCAPE pressed).

Parameter block size : 5

Parameter block format :

XY	=	<LSB of buffer address>
XY+1	=	<MSB of buffer address>
XY+2	=	<max. no. of characters>
XY+3	=	<lowest ASCII code>
XY+4	=	<highest ASCII code>

On exit :

C=0	indicates that the line of input was terminated by a carriage return (ASCII 13)
C=1	indicates that the line of input was terminated by an escape condition
Y=	<length of input line>

Notes

1. The buffer address specified in XY and XY+1 must be in the main memory of the microcomputer, not a co-processor.
2. All input characters (with the exception of ASCII 13, ASCII 21 and ASCII 127 – see below) are sent to the currently selected output stream(s) (as selected by OSBYTE 3); only characters within the range specified in XY+3 and XY+4 are entered into the user buffer.
3. ASCII 13 (RETURN) terminates the OSWORD 0 call.
4. If the buffer is not empty, ASCII 127 (DELETE) is output and deletes the last character placed in the buffer; if the buffer is empty it is ignored and not output.
5. ASCII 21 (CTRL+U) deletes all the characters placed in the buffer and sends one ASCII 127 to the output stream for each character that was in the buffer.
6. If the count of the number of characters input reaches the number specified in XY+2, further characters are ignored and cause ASCII 7 (CTRL+G) to be sent to the output stream, which will usually cause a short "bleep". Note that DELETE and CTRL+U will still function as described above.
7. OSWORD 0 must not be used in a routine to service interrupts or events.

OSWORD 1 (&01) Read system clock

This call returns the current value of the system clock.

Parameter block size : 5

Parameter block unused on entry.

On exit : The parameter block contains the value of the system clock at the instant of the call:

```
XY     = <time LSB>
XY+1   =       .
XY+2   =       .
XY+3   =       .
XY+4   = <time MSB>
```

Note

The system clock may be accessed or reset from BASIC using the pseudo-variable `TIME` (as distinct from `TIME$`, which accesses the CMOS clock – see below).

OSWORD 2 (&02) Write system clock

This call allows the system clock to be reset to a specified value.

Parameter block size : 5

Parameter block format :

```
XY     = <time LSB>
XY+1   =       .
XY+2   =       .
XY+3   =       .
XY+4   = <time MSB>
```

On exit : The parameter block remains unchanged.

See note under OSWORD 1.

OSWORD 3 (&03) Read interval timer

This call returns the current value of the interval timer.

Parameter block size : 5

Parameter block unused on entry.

On exit : The parameter block contains the value of the interval timer at the instant of the call:

XY = <time LSB>
XY+1 = .
XY+2 = .
XY+3 = .
XY+4 = <time MSB>

OSWORD 4 (&04) Write interval timer

This call allows the interval timer to be reset to a specified value.

Parameter block size : 5

Parameter block format :
XY = <time LSB>
XY+1 = .
XY+2 = .
XY+3 = .
XY+4 = <time MSB>

On exit : The parameter block remains unchanged.

Note

To produce repeated interrupts, the routine servicing the timer interrupt should reload the timer with the appropriate number (eg to produce an interrupt every 10 seconds, reload it with −1000 (&FFFFFFFC18).

OSWORD 5 (&05) Read byte from I/O processor memory

This call provides a means of accessing any memory location in the I/O processor.

Parameter block size : 5

Parameter block format :
XY = <LSB of I/O processor memory address>
XY+1 = .
XY+2 = .
XY+3 = <MSB of I/O processor memory address>

On exit : The byte from the specified location in the I/O processor is returned in XY+4.

Notes

1. Parameter block bytes XY+2 and XY+3 need not be specified if the I/O processor memory is fully accessible using a 16 bit address.
2. Whilst this call may be used to access MOS and VDU variables, the use of the appropriate OSBYTE call is recommended.
3. Direct memory access is recommended if speed is an important consideration but only if the program is guaranteed to be running in the I/O processor.

OSWORD 6 (&06) Write byte to I/O processor memory

This call allows a single byte to be stored in the I/O processor memory.

Parameter block size : 5

Parameter block format :

XY	=	<LSB of I/O processor memory address>
XY+1	=	.
XY+2	=	.
XY+3	=	<MSB of I/O processor memory address>
XY+4	=	<byte to be written>

On exit : The parameter block remains unchanged.

See notes under OSWORD 5.

OSWORD 7 (&07) Generate a sound

This call provides the functions necessary to produce simple sounds. More sophisticated sound generation (under the control of a sound envelope) is achieved using this call in conjunction with OSWORD 8 (see below).

The sound circuits have four sound channels, allowing you to produce up to four simultaneous sounds. Each channel has a channel number in the range 0 to 3:

– channel number 0 is the “noise” channel, which produces a (controllable) hissing noise;
– channels 1, 2 and 3 are the tone channels which produce recognisable tones with a musical pitch.

Although the sound channels can be used quite separately (eg to play three different tunes at the same time!), they are usually used in unison to produce a richer, more interesting sound. The parameter block used by OSWORD 7 includes a number to specify which channel should obey it; if that channel is already producing a sound, the command will usually be "queued" in the sound queue for that channel until all sound commands ahead of it in that queue have finished. Each sound channel queue is separate and can hold a maximum of six commands; if the appropriate queue is full, the call to issue the extra sound command will not return until there is space in the queue to insert the new sound command.

Parameter block size : 8

Parameter block format : (see definitions below)

XY: bit 0, bit 1 } <channel number>
bit 2 is unused
bit 3 is unused
bit 4 <flush control bit>
bit 5 is unused
bit 6 is unused
bit 7 is unused

XY+1: bit 0, bit 1 } <synchronisation>
bit 2 is unused
bit 3 is unused
bit 4, bit 5, bit 6, bit 7 } <continuation control>

XY+2 = <LSB of amplitude>
XY+3 = <MSB of amplitude> (currently ignored)
XY+4 = <LSB of pitch>
XY+5 = <MSB of pitch> (currently ignored)
XY+6 = <LSB of duration>
XY+7 = <MSB of duration> (currently ignored)

On exit : The parameter block is unaltered and the specified commands will have been entered into the sound queue for the specified channel (ie actual generation of the sound may not have commenced).

Notes

<channel number> must lie in the range 0 to 3 and identifies which channel is to produce the sound.

<flush control> is either 0 or 1:

0 indicates that the new sound is to be queued, pending completion of any current sound on the specified channel.

1 indicates that the new sound is to be produced immediately, terminating any current sound on the specified channel.

<synchronisation> must lie in the range 0 to 3:

0 indicates that the sound is to be produced as soon as the sound channel is ready (ie when the queue is empty).

1–3 indicates that production of the sound is to be delayed until that number of other sound channels have sound commands with <synchronisation> greater than zero at the head of their queues.

<continuation> must be either 0 or 1:

0 indicates that the sound should be produced when the sound channel is ready for it (ie when the preceding sound has been produced for its specified <duration>).

1 indicates that the preceding sound in the specified channel should be allowed to complete the release phase of its envelope (if one was specified – see below).

Note that the sound command with <continuation> set to 1 is not produced at all.

If <continuation> is greater than 1, the sound command will be passed to the paged ROMs as an unrecognised SOUND command.

<amplitude> is treated as a signed 16 bit integer and should be in the range −15 (&FFF0) to +4 (0010). Values in the range 0 to −15 specify a fixed amplitude (volume) for the sound, with 0 being silent and −15 the loudest. Values in the range 1 to 4 select an "envelope" which dictates how the sound's pitch and loudness should vary during the duration of the sound command.

Note that a further twelve envelopes (5 (&05) to 16 (&10)) may be used if neither cassette nor RS423 output is being used – see OSWORD 8.

<pitch> is in the range 0 to 255.

For the tone channels (1–3) it specifies the pitch of the note in intervals of a quarter of a semitone, with middle C produced by a value of 52.

Note that this pitch may be modulated by a pitch envelope.

For the noise channel (0), <pitch> specifies the tonal quality of the noise; bits 0–2 only are significant:

<pitch>	**Result**
0	High frequency periodic noise
1	Medium frequency periodic noise
2	Low frequency periodic noise
3	Periodic noise with frequency related to pitch of channel 1
4	High frequency "white" noise
5	Medium frequency "white" noise
6	Low frequency "white" noise
7	White noise with frequency related to pitch of channel 1

Periodic noise has a definite pitch, sounding like a roughened version of a tone; "white" noise sounds more like a hissing or roaring.

<duration> is in the range 0 – 255.

A <duration> in the range 0–254 produces a sound of the equivalent number of twentieths of a second (maximum approximately 12.5 seconds). If an envelope has been selected by the <amplitude> parameter, <duration> defines the total time up to the start of the release phase of the envelope (see OSWORD 8).

A <duration> of 255 produces an 'indefinite' note which is terminated only by an escape event or when a note with <flushing> set is sent to the same channel).

OSWORD 8 (&08) Define a sound envelope

The sounds produced by OSWORD 7 have very little dynamic quality – they start and stop, but are otherwise constant in volume, pitch and tonal quality. This call allows you to make sounds much more interesting, by specifying how two important qualities of the sound (pitch and volume) will change during the duration of a sound. This specification, which is called an "envelope", is stored but not used until selected by a positive amplitude parameter in an OSWORD 7 call (see above). You can always store up to four different envelope definitions, although up to 16 envelopes may be defined if neither RS423 nor cassette output is being used.

Note that envelopes define the amplitude of the sound with time, but only modulate the pitch of the sound which is specified in the sound command.

Designing envelopes is a fairly intricate task. The best method is to draw the required envelope on graph paper, convert the drawn shape into envelope parameters, try it out and then adjust the drawing and envelope command until the required effect is obtained. The envelope editor supplied as part of the Welcome software may be used for experimenting with different envelope parameters.

The Amplitude envelope

The amplitude part of the envelope is divided into four consecutive phases; some sounds will require all four components, others merely a subset:

– The attack phase

  The attack phase corresponds to the start of a sound, such as the very sudden build-up in volume of a cymbal crash or piano note, or the slow build-up in volume of an approaching car engine.

– The decay phase

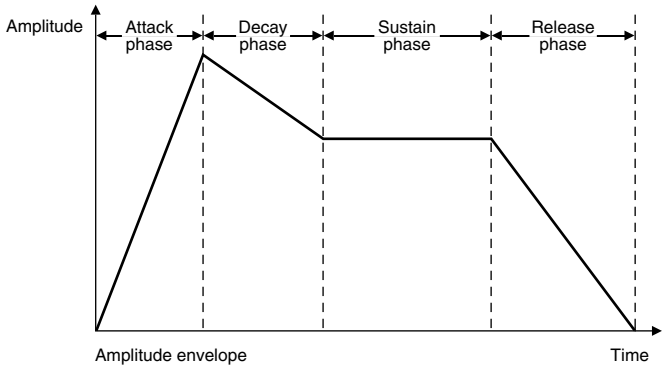
  The decay phase corresponds to the decrease in volume after the initial peak, such as when a piano string settles down.

– The sustain phase

  The sustain phase corresponds to the fairly static period in the life of the sound when the volume changes little or not at all.

– The release phase

  The release phase corresponds to the dying out of a sound, such as when the piano key is released and the damping mechanism rapidly muffles the string's vibration.



The shape of the amplitude envelope is dictated by a combination of factors:

– The attack phase of the amplitude envelope begins with the final amplitude of the previous sound on that channel.

– The total time allowed for the attack, decay and sustain phases of the envelope is set by the <duration> parameter of the OSWORD 7 call using the envelope so that a command with a very short duration can miss out the decay and sustain phases altogether.

– Assuming that the OSWORD 7 <duration> is long enough, the length of the attack and decay phases is determined by their <amplitude change rates> and <target levels>, as these phases end when the amplitude has reached the target level for that phase.

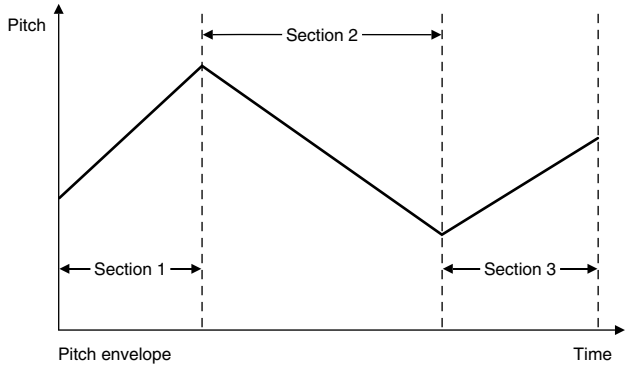
– The sustain phase continues until either the sound has been produced for its specified OSWORD 7 <duration> parameter, or until the amplitude has reached 0.

– The release phase usually begins only if there are no more sound commands to execute in the queue for that channel and continues until either another sound command is received for that channel or the amplitude reaches 0. If the next sound command in the queue has its <continuation> bit set to 1, the release phase will always continue until the amplitude reaches 0.

  The amplitude envelope has a resolution of 128 steps (0 to 127) as opposed to the 16 steps (0 to −15) of the <amplitude> parameter used in OSWORD 7 calls. The maximum and minimum volumes available with both are the same – the envelope simply uses finer steps, one eighth of the size of the steps used in the sound command.

The Pitch envelope

The pitch part of the envelope is divided into three sections, whose length is determined directly by the <pitch length n> parameters (as defined below).



The pitch envelope has the same resolution and range as the <pitch> parameter used in OSWORD 7 calls. If the result of adding the pitch envelope to the sound command <pitch> is outside this range, the pitch actually heard is reduced MOD 256.

Parameter block size : 14

Parameter block format : (see definitions below)

XY = <envelope number>
XY+1: bits 0–6 = <step length>
bit 7 = <pitch envelope repeat>
XY+2 = <pitch change rate 1>
XY+3 = <pitch change rate 2>
XY+4 = <pitch change rate 3>
XY+5 = <pitch length 1>
XY+6 = <pitch length 2>
XY+7 = <pitch length 3>
XY+8 = <attack amplitude change rate>
XY+9 = <decay amplitude change rate>
XY+10 = <sustain amplitude change rate>
XY+11 = <release amplitude change rate>
XY+12 = <attack target level>
XY+13 = <decay target level>

On exit : The parameters associated with the specified envelope number will have been stored for use with subsequent OSWORD 7 calls.

Notes

OSWORD 8 is the call implemented by the BASIC `ENVELOPE` command and the parameters defined below apply equally to both uses.

<envelope number> may be in the range 1 to 4 (or 1 to 16 – see above).

This specifies the number of the envelope being (re)defined.

<step length> must be in the range 1 to 127.

This defines the number of hundredths of a second used for each step in the envelope definition.

<pitch envelope repeat> may be 0 or 1:

0 means repeat the pitch envelope as many times as necessary if it finishes before the sound does.

1 means do not repeat the pitch envelope – if the sound lasts longer than the pitch envelope, the pitch setting at the end of the pitch envelope will be maintained until the sound finishes

<pitch change rate 1> is interpreted as an eight bit signed integer and may be in the range −128 to +127. This defines the rate at which the pitch should change (in pitch units of quarter of a semitone) per <step length> in section 1 of the pitch envelope.

<pitch change rate 2> is as <pitch change rate 1>, but for section 2 of the pitch envelope.

<pitch change rate 3> is as <pitch change rate 1>, but for section 3 of the pitch envelope.

<pitch length 1> is in the range 0 to 255.

This defines the length of section 1 of the pitch envelope, in units of <step length>.

<pitch length 2> is as <pitch section 1 length>, but for section 2.

<pitch length 3> is as <pitch section 1 length>, but for section 3.

<attack amplitude change rate> is interpreted as an eight bit signed integer, in the range −127 to +127.

This defines the rate at which the amplitude changes per <step length> during the attack phase. Note that the units are one eighth of the <amplitude> units used in OSWORD 7 calls.

<decay amplitude change rate> is as for <attack amplitude change rate>, but defines the rate used in the decay phase.

<sustain amplitude change rate> is interpreted as an eight bit signed integer, with an allowed range of 0 to −127.

This defines the rate at which the amplitude changes during the sustain phase.

<release amplitude change rate> is interpreted as an eight bit signed integer, with an allowed range of 0 to −127.

This defines the rate at which the amplitude is reduced during the release phase.

<attack target level> should be in the range 0 to 126.

This defines the amplitude that must be reached to end the attack phase.

<decay target level> is as <attack target level>, but defines the end of the decay phase.

OSWORD 9 (&09) Read pixel logical colour

This call is used to determine the logical colour of the pixel at given co-ordinates on the graphics screen.

Parameter block size : 5

Parameter block format :
XY = <LSB of X co-ordinate>
XY+1 = <MSB of X co-ordinate>
XY+2 = <LSB of Y co-ordinate>
XY+3 = <MSB of Y co-ordinate>

On exit : XY+4 contains the logical colour of the pixel specified.

If XY+4= &FF then the pixel was outside the current graphics window or the screen is in a non-graphics mode.

Notes

1. The co-ordinates should be specified relative to the current graphics origin, if it has been redefined using a VDU 29 command – see section E.
2. This call must not be used in a routine used to service interrupts or events.

OSWORD 10 (&0A) Read a character definition

The characters displayed in all modes other than 7 and 135 are defined as an 8×8 matrix of dots. This call provides a means of reading the definition for a specified ASCII code.

Parameter block size : 9

Parameter block format : XY = <ASCII code of character required>

On exit : Bytes XY+1 to XY+8 contain the definition of the specified character:

XY+1 = <top row of definition>
.
.
XY+8 = <bottom row of definition>

Notes

1. Bits set in each row of the character definition will be displayed in the current text foreground colour; bits clear in each row will be displayed in the current text background colour. See also VDU command 5 in section E.
2. The definitions returned for ASCII codes 0 to 31 and 127 (ie non-printing characters) are not significant.
3. This call must not be used in a routine used to service interrupts or events.

OSWORD 11 (&0B) Read the palette

The VDU palette is a block of 64 bits of RAM in the video ULA which determines the displayed (physical) colour of each logical colour.

This call provides a means of determining the physical colour associated with a particular logical colour.

Parameter block size : 5

Parameter block format : XY = <logical colour to read>

On exit : XY+1 contains the physical colour associated with the specified logical colour.
XY+2 = 0
XY+3 = 0
XY+4 = 0

OSWORD 12 (&0C) Write the palette

This call allows the physical colour associated with a particular logical colour to be changed.

Parameter block size : 5

Parameter block format : XY = <logical colour to change>
XY+1 = <new physical colour>
XY+2 = 0
XY+3 = 0
XY+4 = 0

On exit : The parameter block is unchanged.

Note

OSWORD 12 duplicates the function of VDU command 19 (see section E). However, the OSWORD call is faster and may be used in interrupt routines.

OSWORD 13 (&0D) Read current and previous graphics cursor positions

The VDU driver stores the last two positions of the graphics cursor to use when plotting shapes. This call provides a means of reading these positions.

Parameter block size : 8

Parameter block unused on entry

On exit : XY = <LSB of previous X coordinate>
XY+1 = <MSB of previous X coordinate>
XY+2 = <LSB of previous Y coordinate>
XY+3 = <MSB of previous Y coordinate>
XY+4 = <LSB of current X coordinate>
XY+5 = <MSB of current X coordinate>
XY+6 = <LSB of current Y coordinate>
XY+7 = <MSB of current Y coordinate>

OSWORD 14 (&0E) Read CMOS clock

OSWORD 14 provides three different read functions associated with the CMOS clock:

1. Read clock in string format
2. Read clock in Binary Coded Decimal (BCD) format
3. Convert BCD clock value into string format

1. Read clock in string format

Parameter block size : 25

Parameter block format : XY = 0

On exit : The parameter block contains a 24-byte character string in the form:

ddd,nn mmm yyyy.hh:mm:ss
where:

ddd	is a three character abbreviation for the day
nn	is the day number
mmm	is a three character abbreviation for the month
yyyy	is the year
hh	is the hour (in 24-hr clock notation)
mm	is the number of minute past the hour
ss	is the number of seconds

XY+24 contains a carriage return character (&0D)

2. Read clock in BCD format

Parameter block size : 7

Parameter block format : XY = 1

On exit : The parameter block contains the 7-byte BCD clock value:

XY	= <year>	(00–99)
XY+1	= <month>	(01–12)
XY+2	= <day of month>	(01–31)
XY+3	= <day of week>	(01–07)
XY+4	= <hours>	(00–23)
XY+5	= <mins>	(00–59)
XY+6	= <secs>	(00–59)

For the day of the week, Sunday = 01, Monday = 02 etc.

3. Convert BCD clock value into string format

Parameter block size : 25

Parameter block format : XY = 2

XY+1, XY+2, XY+3, XY+4, XY+5, XY+6, XY+7 contain the BCD clock value to be converted

On exit : The parameter block contains the 24-byte clock string (as defined under option 1, above).

Notes

1. OSWORD 14 with XY=2 will not work with certain early external 6502 second processors.
2. Interrupts are enabled during this call.

OSWORD 15 (&0F) Write CMOS clock

OSWORD 15 provides three write functions associated with the CMOS clock.

1. Change the time only
2. Change the date only
3. Change the date and time

1. Change the time only

Parameter block size : 9

Parameter block format :
- XY = 8
- XY+1 = <ASCII code for first hours digit>
- XY+2 = <ASCII code for second hours digit>
- XY+3 = 58 (ie ASCII code for **:**)
- XY+4 = <ASCII code for first minutes digit>
- XY+5 = <ASCII code for second minutes digit>
- XY+6 = 58
- XY+7 = <ASCII code for first seconds digit>
- XY+8 = <ASCII code for second seconds digit>

On exit : The parameter block remains unchanged.

2. Change the date only

Parameter block size : 16

Parameter block format :
- XY = 15
- XY+1 = <ASCII code for first day character>
- XY+2 = <ASCII code for second day character>
- XY+3 = <ASCII code for third day character>
- XY+4 = 44 (ie ASCII code for **,**)

XY+5 = <ASCII code for first day digit>
XY+6 = <ASCII code for second day digit>
XY+7 = 32 (ie ASCII code for space)
XY+8 = <ASCII code for first month character>
XY+9 = <ASCII code for second month character>
XY+10 = <ASCII code for third month character>
XY+11 = 32
XY+12 = <ASCII code for first year digit>
XY+13 = <ASCII code for second year digit>
XY+14 = <ASCII code for third year digit>
XY+15 = <ASCII code for fourth year digit>

On exit : The parameter block is unchanged

3. Change date and time

Parameter block size : 25

Parameter block format : XY = 24
XY+1 – XY+15 = <date string> (as in 2, above)
XY+16 = 46 (ASCII code for .)
XY+17 – XY+24 = <time string> (as in 1, above)

On exit : The parameter block is unchanged

D.4 Passing commands to the MOS (OSCLI)

OSCLI is used by high level languages and utilities to process any commands that start with an asterisk. It therefore provides consistent access to MOS commands, filing system commands and other commands from any language.

Languages may also provide their own keywords to simplify the passing of variable data to OSCLI, such as the BASIC keyword `OSCLI`.

Call address &FFF7
Indirected via &208 (CLIV)

OSCLI uses the MOS Command Line Interpreter (the CLI). The CLI may be selected as the current language, as a result of errors such as disabling all languages (using `*UNPLUG`) then resetting the system, or booting a disc which has no !BOOT file when using a second processor. You may also select the CLI deliberately, by executing the MOS command `*GO` without a destination address.

In assembly language programs, MOS commands may be executed by calling the corresponding routine directly (if documented), or by storing the text of the command in RAM and passing its address to OSCLI in the X and Y registers, as described below.

On entry : X=<LSB of command string start address>
Y=<MSB of command string start address>

The command string itself must terminate with a carriage return character (ASCII 13).

On exit : All registers are undefined
C, N, V and Z are undefined
The interrupt status is preserved

Notes

1. OSCLI ignores leading asterisks and spaces, and will ignore the command text if it is more than 255 characters in length or starts with `*¦`.
2. Interrupts may be enabled during the call.

D.5 Reading from the input stream (OSRDCH)

OSRDCH is used by the MOS, Filing Systems and languages to read a character from the currently selected input stream (ie either the keyboard or RS423). The keyboard is selected by default and selection of either stream is made using OSBYTE 2/`*FX2`.

The routine also performs the necessary manipulation to implement features such as soft key expansion and cursor editing.

Call address &FFE0
Indirected via &210 (RDCHV)

To read a number of characters from the input stream into memory, it is probably more convenient to use OSWORD 0 (which uses OSRDCH !)

There are no entry parameters

On exit : The carry flag (C) indicates the validity of the character retrieved from the input buffer:

C=0 indicates the code is valid – the ASCII code is returned in A.
C=1 indicates an error – A contains a number indicating the type of error, for example, if A=27 (&1B) then an escape condition has occurred (and which would be cleared by OSBYTE 124 or OSBYTE 126).

X and Y are preserved
The interrupt status is preserved.

Note

Interrupts may be enabled during the call.

D.6 Reading from the screen / paged ROMs (OSRDSC)

OSRDSC is a dual-purpose routine which will access a location in either the screen memory or a paged ROM.

Call address &FFB9
Not indirected

On entry : To read a byte from the current screen memory:

location &F6 = <LSB of location required>
location &F7 = <MSB of location required>

To read a byte from a paged ROM:

Y=<ROM number>

location &F6 = <LSB of location required>
location &F7 = <MSB of location required>

(resultant address being in the range &8000–&BFFF)

On exit : The byte read is returned in A.

Notes

1. This call cannot be used from a co-processor.
2. Some authors have used the name OSRDRM to identify the use of this routine to access bytes in paged ROMs.

D.7 Writing to the output stream (OSWRCH, OSNEWL, OSASCI)

OSWRCH, OSNEWL and OSASCI are used by the MOS, Filing Systems and languages to write ASCII characters to the current output stream (ie one or more of the display, printer, RS423 port or a `*SPOOL` file). The selection of stream(s) is made using OSBYTE 3/`*FX3`.

D.7.1 OSWRCH

OSWRCH is the main entry point for all output.

Call address &FFEE
Indirected via &20E (WRCHV)

On entry : A=<ASCII code of character to be written>

On exit : A, X and Y are preserved
C, N, V and Z are undefined
The interrupt status is preserved

Notes

1. The characters sent to OSWRCH may be output, trapped, ignored (printer ignore character) or translated into commands (eg VDU driver), etc, depending on the output streams selected.
2. Interrupts may be enabled during the call.

D.7.2 OSNEWL

OSNEWL sends a 'new line' sequence to the current output stream(s) (ie LF (ASCII 10) followed by CR (ASCII 13)).

Call address &FFE7
Not indirected

No entry parameters

On exit : A=13
X and Y are preserved
C, N, V and Z are undefined
The interrupt status is preserved.

Notes

1. OSNEWL uses OSWRCH.
2. Interrupts may be enabled during the call.

D.7.3 OSASCI

OSASCI combines the functions of OSWRCH and OSNEWL, ie it is equivalent to OSWRCH except when the character to be sent is ASCII 13, when it is equivalent to OSNEWL.

Call address &FFE3
Not indirected

On entry : A=<ASCII code of character to be written>

On exit : A, X and Y are preserved
C, N, V and Z are undefined
The interrupt status is preserved

Note

Interrupts may be enabled during the call.

D.8 Writing to the screen memory (OSWRSC)

This call provides a write equivalent to OSRDSC (with the obvious non-availability of paged ROM access).

Call address &FFB3
Not indirected

On entry : location &D6=<LSB of address to be written>
location &D7=<MSB of address to be written>
A=<value to be written>

On exit : A, X and Y are preserved.

Note

This call cannot be used from a co-processor.

D.9 Generating events (OSEVEN)

Events are conditions detected by the MOS during interrupts, such as an input buffer becoming full or an RS423 error. The user can choose to handle events after the MOS has done its normal processing, by enabling the event using OSBYTE 14 and setting the EVENTV vector (&220–1) to point to the routine required.

OSEVEN generates an event (ie simulates the condition which would normally cause the event).

Call address &FFBF
Not indirected

On entry : Y=<event number>

If the event has been enabled with OSBYTE 14/`*FX14`, the service routine pointed to by EVNTV will be entered with X preserved and:

A=<event number>
Y=<content of the accumulator>

Notes

1. To generate an event and have it handled even if it is not enabled, load A with the event number and X and Y with the desired parameters, disable interrupts and indirect through the event vector.
2. This call will not function from a co-processor.

D.10 Generalised string input (GSINIT and GSREAD)

The MOS uses GSINIT and GSREAD to process strings used in commands such as `*KEY` and `*LOAD`. The main advantage of using these calls is that the character ¦ may be used to introduce control characters which would otherwise be difficult to type directly from the keyboard.

D.10.1 GSINIT

This routine should be used to set up an information byte (in page zero) for a string which is to be used for input using GSREAD (See below).

Call address &FFC2
Not indirected

On entry : location &F2=<LSB of string address>
location &F3=<MSB of string address>

Y=<offset from address in &F2/&F3> (if required)

The string may be terminated by one of:

– the first space character;
– the first carriage return character;
– the second quotation mark character.

according to the value of C:

C=0 specifies any of the above characters
C=1 specifies carriage return or the second quotation mark only.

On exit : Y=<offset of first non-space character>
A=<contains the first non-space character>
Z=0

Z=1 if the string is null.

Note

This call should not be used from a co-processor.

D.10.2 GSREAD

This routine returns a character from a string initialised by GSINIT ie it must be used only after a GSINIT call.

On entry : locations &F2 and &F3 must be unchanged from the last GSINIT call.

Y should be the value returned from GSINIT (or a previous GSREAD call).

On exit : A=<character read from the string>
X is preserved
Y=<offset of next character to be read>
C=0

C=1 indicates that the end of the string has been reached.

Note

This call should not be used from a co-processor.

E The VDU driver

E.1 Introduction to the VDU driver

The VDU driver is a complex set of routines used by the MOS to display all text and graphical output. It also provides one of the routes for output to the printer.

The VDU driver is used by selecting the VDU as one of the output streams (using OSBYTE 3/`*FX3`); all subsequent character output will then be routed to it (as well as to any other selected destinations) by OSWRCH.

Provided that the VDU driver is active, it treats the bytes it receives either as characters to be displayed or as "VDU commands" – instructions which tell it to perform specific functions:

ASCII codes	Interpretation in modes 0–6 (and modes 128–134)	Interpretation in mode 7 (and mode 135)
0 – 31	VDU commands	VDU commands
32 – 126	Characters to be displayed	Characters to be displayed
127	Backspace and delete	Backspace and delete
128 – 159	Characters to be displayed	Teletext control codes
160 – 255	Characters to be displayed	Characters to be displayed

VDU commands

Some VDU commands stand alone, others require parameters such as a display mode, a colour or the co-ordinates of a point on the screen, as given in the descriptions of the commands in section E.3.

The number of parameters required by each command is fixed so that the VDU driver can tell immediately whether parameters are required to complete it. If parameters are required, bytes received by the VDU driver immediately after the command are queued until sufficient parameters have been read. Note that these parameters have no other meaning to the VDU driver, i.e. they are not interpreted as VDU commands nor displayed as text, only used as data for the current VDU command.

The process of splitting bytes sent to the VDU driver into commands and characters to be displayed is referred to as *parsing*.

When the command is complete, it is executed immediately. The number of bytes necessary to complete the current command may be ascertained using OSBYTE 218/&DA.

Characters to be displayed

Whenever the VDU driver receives an ASCII code corresponding to a character to be displayed, it merely writes the appropriate character pattern to the display memory, causing it to appear on the screen:

– In the case of modes 0 – 6 (and 128 – 134) each character is composed of a pattern of 8 × 8 pixels, stored as a sequence of 8 bytes:

– In the case of modes 7 (and 135), it is the ASCII code (i.e. a single byte), which is written to the display memory – the character or symbol which appears on the screen is generated by the teletext display chip (SAA5050).

  The teletext control codes (128 – 159) are also written to the screen memory as single bytes and are used by the teletext display circuitry to determine the interpretation of the remaining characters on the same screen line.

Using VDU commands

As described above, VDU commands are identified by a number in the range 0 to 31 and the individual commands may be issued in a variety of different ways:

– from assembly language programs

  VDU commands are issued by passing the command number and the necessary parameters to the VDU driver using OSWRCH (i.e. in exactly the same way as for writing 'printable' characters).

  Certain commands (particularly those relating to graphics co-ordinates) require parameters which lie outside the range 0 – 255. In these cases, the parameter must be sent as a two-byte pair, LSB first, MSB second.

– from BASIC programs

  Many frequently-used VDU commands have equivalent BASIC keywords (like `CLS`, `MOVE`, `DRAW`, `PLOT` etc, as indicated in the descriptions which follow); others are issued directly by means of `VDU` statements.

  All values following the keyword `VDU` are directed to the VDU driver as either a single byte (if followed by a comma) or as a two-byte pair (if followed by a semi-colon).

Note that the punctuation is part of the syntax of the BASIC language and not a requirement of the VDU driver.

Note also that the BASIC language allows the use of the ¦ symbol in VDU statements. ¦ is equivalent to 0,0,0,0,0,0,0,0,0 and, since no VDU command takes more than nine parameters, will therefore complete any VDU command sequence. (Any excess values are equivalent to VDU 0 commands, which have no effect).

`VDU` statements may, of course, be executed immediately in BASIC's command mode.

– in the command mode of the current language

Most languages display their output using the VDU driver and, where this is the case, it is possible to issue VDU commands using CTRL in combination with other keys on the keyboard. Note. however, that not all commands are appropriate to all languages and that care is needed with specification of parameters.

The key (used in conjunction with CTRL) for a particular VDU command is given in the appropriate command description.

Not all VDU commands have an effect in all the display modes.

The teletext display modes (7 and 135) work quite differently to the other modes and as a result many of the VDU commands will not work in these modes, and are ignored by the VDU driver. The commands which will work in teletext modes are described in section E.5.

Graphics or graphics-dependent commands cannot be used in modes 3 (131) and 6 (134) – the 'text-only modes'.

E.2 VDU command summary

A full list of VDU commands is given below in 'functional' groups. Descriptions of the various commands are given in section E.3.

Colour selection

dec.	hex.	description
17	11	Define text colour
18	12	Define graphics colour
19	13	Define logical colour
20	14	Restore default logical colours

Cursor movement

dec.	hex.	description
8	08	Back space
9	09	Forward space
10	0A	Next line
11	0B	Previous line
13	0D	Carriage return
30	1E	Home cursor
31	1F	Tab cursor
127	7F	Backspace and delete

Graphics

dec.	hex.	description
25	19	PLOT commands (various)
29	1D	Define graphics origin

Miscellaneous functions

dec.	hex.	description
0	00	no effect
7	07	Produce BELL sound
23	17	Various effects
27	1B	no effect

Mode selection

dec.	hex.	description
22	16	Select screen mode

Printer control

dec.	hex.	description
1	01	Send next byte to printer only
2	02	Enable printer
3	03	Disable printer

Scrolling

dec.	hex.	description
14	0E	Enable paged mode
15	0F	Disable paged mode

Text positioning

dec.	hex.	description
4	04	Print text at text cursor
5	05	Print text at graphics cursor

VDU driver control

dec.	hex.	description
6	06	Enable VDU driver
21	15	Disable VDU driver

Windows

dec.	hex.	description
12	0C	Clear text window
16	10	Clear graphics window
24	18	Define graphics window
26	1A	Restore default windows
28	1C	Define text window

E.3 VDU command descriptions

Each VDU command is described below, the sequence being numerical order of command number. Refer to the functional listing in section E.2 for information on commands related to a specific topic.

The information provided for each call is presented as follows:

– *Command number and function*

  The command number is specified in decimal (and hexadecimal); the function is a brief statement of the purpose of the command;

– *Description*

  a description of the effect of the command;

– *Number of parameters*

  This entry specifies the number of parameters associated with the command;

– *Parameter specification*

  This entry specifies the range of values permissible for the command parameters (if any);

– *Keyboard equivalent*

  Where appropriate, this entry specifies the keystrokes necessary to invoke the command in the command mode of the current language – further keystrokes will be required if the command requires parameters – these will not be displayed;

– *BASIC equivalent*

  Where appropriate, this entry specifies the BASIC keyword and parameters equivalent to the VDU command.

If necessary, further information is provided under the heading *Notes*.

VDU 0 (&00) Null

This command has no effect whatsoever, beyond taking a small and undefined amount of time.

Number of parameters : 0

Keyboard equivalent : CTRL + @

VDU 1 (&01) Send next character to printer only

This causes the next character to be sent to the printer only, provided the printer has been enabled by VDU 2.

Number of parameters : 1

Parameter specification : <0–255> interpreted as an ASCII code

Keyboard equivalent : CTRL + A

VDU 2 (&02) Enable printer

VDU 2 causes all subsequent printable characters (ASCII codes 32 to 126) and ASCII codes 8 – 13 sent to the VDU drivers to be sent to the printer as well as to the screen.

Number of parameters : 0

Keyboard equivalent : CTRL + B

Notes

1. If printing is enabled using VDU 2, non-printing characters may still be sent to the printer using VDU 1 (as described above).
2. Characters are sent to the printer regardless of whether the VDU drivers are enabled (using VDU 6) or disabled (using VDU 21) – see below.

VDU 3 (&03) Disable printer

VDU 3 reverses the effect of VDU 2.

Number of parameters : 0

Keyboard equivalent : CTRL + C

Note

After a VDU 3 command, characters cannot be sent to the printer using VDU 1 (as described above).

VDU 4 (&04) Print at text cursor

VDU 4 causes printable characters to be printed at the current text cursor position using the current text foreground and background colours.

Number of parameters : 0

Keyboard equivalent : CTRL + D

Notes

1. The text cursor will be displayed provided that it has not been disabled using VDU 23 1.
2. After the character has been printed, the text cursor is normally moved one character position to the right (and to the start of the next screen line if necessary). The direction of the cursor movement may be changed using VDU 23 16 (see below).

VDU 5 (&05) Print text at graphics cursor

VDU 5 causes printable characters to be printed at the current graphics cursor position. Characters are displayed in the current graphics foreground colour; the background colour is left unaltered.

Number of parameters : 0

Keyboard equivalent : CTRL + E

Notes

1. This command has no effect in text-only or teletext modes.
2. After the character has been printed, the graphics cursor is normally moved one character position to the right (to the start of a new line in the graphics window if necessary, or to the top left corner of the graphics window if there are no more lines). The direction of cursor movement may be changed using VDU 23 16 (see below).
3. Printing text at graphics co-ordinates is considerably slower than printing at normal text co-ordinates.
4. The characters are printed so that the top-left corner of the character occupies the current graphics cursor position.

VDU 6 (&06) Enable VDU driver

VDU 6 restores the functions of the VDU driver after it has been disabled (using VDU 21).

Number of parameters : 0

Keyboard equivalent : CTRL + F

VDU 7 (&07) Produce BELL sound

VDU 7 generates either the default BELL sound (as specified by `*CONFIGURE LOUD/QUIET` or the BELL sound defined following the use of OSBYTE/`*FX 211–214`.

Number of parameters : 0

Keyboard equivalent : CTRL + G

VDU 8 (&08) Back space

VDU 8 causes either the text cursor (by default or after a VDU 4 command) or the graphics cursor (after a VDU 5 command) to be moved one character position in the negative X direction.

Number of parameters : 0

Keyboard equivalent : CTRL + H

Notes

1. The negative X direction is normally left, although it may be changed using VDU 23 16 (see below).
2. In VDU 4 mode if there is a pending carriage return/line feed at the time, it is destroyed and the cursor does not move.

VDU 9 (&09) Forward space

VDU 9 causes either the text cursor (by default or after a VDU 4 command) or the graphics cursor (after a VDU 5 command) to be moved one character position in the positive X direction.

Number of parameters : 0

Keyboard equivalent : CTRL + I

Notes

1. The positive X direction is normally right, although it may be changed using VDU 23 16 (see below).
2. In VDU 4 mode if there is a pending carriage return/line feed, it is executed.

VDU 10 (&0A) Next line

VDU 10 causes either the text cursor (by default or after a VDU 4 command) or the graphics cursor (after a VDU 5 command) to be moved one line in the positive Y direction.

Number of parameters : 0

Keyboard equivalent : CTRL + J

Note

The positive Y direction is normally down but it may be changed using VDU 23 16 (see below).

VDU 11 (&0B) Previous line

VDU 11 causes either the text cursor (by default or after a VDU 4 command) or the graphics cursor (after a VDU 5 command) to be moved one line in the negative Y direction.

Number of parameters : 0

Keyboard equivalent : CTRL + K

Note

The negative Y direction is normally up but it may be changed using VDU 23 16 (see below).

VDU 12 (&0C) Clear text window

VDU 12 clears either the current text window (by default or after a VDU 4 command) or the current graphics window (after a VDU 5 command) to the current text or graphics background colour respectively.

Number of parameters : 0

Keyboard equivalent : CTRL + L

BASIC equivalent : `CLS`

Notes

1. If there is a pending cursor movement, it is destroyed without actually occurring.
2. After a VDU 12 command, the text/graphics cursor is moved to its 'home' position (see VDU 30 below).

VDU 13 (&0D) Carriage return

VDU 13 causes the text cursor (by default or after a VDU 4 command) or the graphics cursor (after a VDU 5 command) to be moved to the negative X edge of the relevant window.

Number of parameters : 0

Keyboard equivalent : CTRL + M

Notes

1. In VDU 4 mode if there is a pending carriage return/line feed at the time, it is destroyed.
2. The negative X edge of the current text or graphics window is normally the left edge but it may be changed using VDU 23 16 (see below).

VDU 14 (&0E) Page mode on

This causes the screen display to wait for SHIFT to be pressed before the next scroll and periodically thereafter.

Number of parameters : 0

Keyboard equivalent : CTRL + N

Notes

1. CAPS LOCK and SHIFT LOCK indicators are illuminated simultaneously while the display is being held pending depression of SHIFT.
2. Scrolling under the effect of VDU 14 normally allows output of approximately 75% of the number of lines in the current window.

VDU 15 (&0F) Page mode off

VDU 15 cancels the effect of VDU 14.

Number of parameters : 0

Keyboard equivalent : CTRL + O

Note

After a VDU 15 command, scrolling is unrestricted although the screen may be 'frozen' using CTRL + SHIFT.

VDU 16 (&10) Clear graphics window

VDU 16 clears the current graphics window to the current graphics background colour.

Number of parameters : 0

Keyboard equivalent : CTRL + P

BASIC equivalent : `CLG`

VDU 17 (&11) Define text colour

VDU 17 is used to assign a logical colour to either the text foreground or text background colour.

Number of parameters : 1

Parameter specification : <0–127> to set the text foreground colour
<128–255> to set the text background colour

If the absolute value of the parameter lies outside the allowed set for the current mode, it is reduced MOD (number of available colours).

Keyboard equivalent : CTRL + Q

BASIC equivalent : `COLOUR` <0–127/128–255>

VDU 18 (&12) Define graphics colour

VDU 18 is used to define either the graphics foreground colour or the graphics background colour and the way in which it is to be plotted on the screen.

Number of parameters : 2

Parameter specification : <plot mode> <colour>

<plot mode> determines the type of colour and the way in which it is to be plotted. For 'pure' colours, the permissible values are:

0 Overwrite the colour on the screen with <colour>
1 OR the colour on the screen with <colour>
2 AND the colour on the screen with <colour>
3 EOR the colour on the screen with <colour>
4 Invert the colour on the screen (i.e. replace the colour on the screen with the colour generated by EORing its logical colour number with (number of logical colours)−1)
5 Leave the colour on the screen unchanged

<colour> is either in the range 0 – 127, in which case it specifies the graphics foreground colour, or in the range 128 – 255, which specifies the graphics background colour as logical colour <colour>−128. In each case, values are reduced MOD (number of colours available in the current mode).

VDU 18 may also be used to select any of the four Extended Colour Fill (ECF) patterns. This is achieved by setting <plot mode> as described below (with n defining the way in which the pattern is to be plotted, as above).

16+n ECF pattern 1
32+n ECF pattern 2
48+n ECF pattern 3
64+n ECF pattern 4

Where an ECF pattern is specified, <colour> = 0 selects the pattern as the graphics foreground colour and <colour> = 128 selects the pattern as the graphics background colour.

Keyboard equivalent : CTRL + R

BASIC equivalent : GCOL <plot mode> <colour>

Note

Other values of <plot mode> have undefined effects.

VDU 19 (&13) Define logical colour

VDU 19 causes a specified logical colour to be represented by a specified physical colour.

Number of parameters : 5

Parameter specification : <logical colour> <physical colour> 0 0 0

<logical colour> must lie within the range of logical colour numbers for the current mode.

<physical colour> is one of:

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

Keyboard equivalent : CTRL + S

Notes

1. <logical colour> is reduced MOD (number of colours) if it lies outside the range for the current mode.

2. <physical colour> is reduced MOD 16 if it lies outside the range 0–15.
3. The remaining three parameters are currently unused and are reserved for future expansions. They must, however, be specified in order to complete a VDU 19 command.

VDU 20 (&14) Restore default logical colours

VDU 20 causes the logical-physical colour map to be reset to its default state for the current mode.

Number of parameters : 0

Keyboard equivalent : CTRL + T

Notes

(See table under VDU 19 for a complete list of physical colours).

1. In 2-colour modes (0, 3, 4, 6, 128, 131, 132 and 134) the default assignments are:

logical colour	physical colour
0	0 (black)
1	7 (white)

In 4-colour modes (1, 5, 128 and 133) the default assignments are:

logical colour	physical colour
0	0 (black)
1	1 (red)
2	3 (yellow)
3	7 (white)

In 16-colour modes (2 and 130), each logical colour is assigned the corresponding physical colour.

2. VDU 20 also resets the text and graphics background colour to Black (colour 0) and the text and graphics foreground colour to White (i.e. colour 1 in 2-colour modes, colour 3 in 4-colour modes, and colour 7 in 16-colour modes).

VDU 21 (&15) Disable VDU driver

VDU 21 prevents the VDU driver performing any of its normal functions until a VDU 6 is received.

Number of parameters : 0

Keyboard equivalent : none (see note 3)

Notes

1. The normal queuing of the stream of characters sent to the VDU driver continues after VDU 21, with sequences of characters being generated as usual. Thus, for example, the sequence VDU 22,6 will not re-enable the VDU driver.
2. Characters may still be sent to the printer after VDU 21.
3. CTRL+U issued from the keyboard has the effect of deleting the current line.

VDU 22 (&16) Select screen mode

VDU 22 causes the screen mode to be set to a specified mode (either normal or shadow as required).

Number of parameters : 1

Parameter specification : <0–7> to select normal screen modes
<128–135> to select shadow screen modes

Keyboard equivalent : CTRL+V

BASIC equivalent : `MODE` <0–7/128–135>

Notes

VDU 22 resets large amounts of the VDU system, as described below. Note, however, that unlike BASICs `MODE` statement, VDU 22 does NOT reset the value of pseudo-variable HIMEM and should therefore be used with caution in BASIC programs.

1. The 6845 CRTC is completely reprogrammed – this is the point at which the effects of `*TV` or `*FX144` occur.

2. Cursor editing is terminated if currently in use.
3. VDU 4 mode is entered – see the description of VDU 4.
4. The text and graphics windows are restored to their default values, the text cursor moved to its "home" position and the graphics cursor to (0,0), as for VDU 26.
5. Paged mode is terminated if currently in use, as for VDU 15.
6. The logical-physical colour map is set to its default for the new mode, as for VDU 20.
7. The ECF patterns are set to their defaults for the new mode (see VDU 23 11 below).
8. The dot pattern for dotted lines is reset to &AA (i.e. dot-space repeated), and the current pattern is also reset to this value (see VDU 23 6 below).
9. The screen is cleared in the same way as for VDU 12.

VDU 23 (&17) Various functions

VDU 23 is a multi-purpose command taking nine parameters, of which the first identifies a particular function. Each of the available functions is described below. Eight additional parameters are required in each case.

VDU 23 0 Control 6845 CRTC directly

VDU23 0 writes a value to one of the 18 6845 registers.

Parameter specification : <register> <value> 0 0 0 0 0 0

<register> is in the range 0 – 17 and identifies a 6845 register. <value> is the required register content.

Users should refer to the Master Series Advanced Reference Manual if they wish to change the contents of these registers.

VDU 23 1 Turn cursor on/off

VDU 23 1 controls the appearance of the cursor on the screen.

Parameter specification : <n> 0 0 0 0 0 0 0

The effect is determined by the value of <n>:

0 stops the cursor appearing.

1 causes the cursor to re-appear.

2 causes the cursor to become steady.

3 causes the cursor to flash slowly (approx 1.5 times per second).

Note

The cursor will always re-appear if cursor editing is entered.

VDU 23 2–5 Set ECF patterns

VDU commands 23 2 to VDU 23 5 are used to define the four Extended Colour Fill patterns:

VDU 23 2 sets ECF pattern 1
VDU 23 3 sets ECF pattern 2
VDU 23 4 sets ECF pattern 3
VDU 23 5 sets ECF pattern 4

Parameter specification : <n1> <n2> <n3> <n4> <n5> <n6> <n7> <n8>

Each of the integers <n1> to <n8> define one row of the pattern, <n1> being the top row, <n8> being the bottom. The logical colours of the pixels in each row depend upon the number of colours available in the current mode.

If the bit settings in each of <n1> to <n8> are denoted by 76543210, then the logical colours of the pixels in each row (from left to right) are:

in 2-colour modes 7,6,5,4,3,2,1,0

in 4-colour modes 73,62,51,40

in 16-colour modes 7531,6420

VDU 23 6 Set dotted lines pattern

VDU 23 6 sets the dot pattern used by dotted line PLOT commands (see VDU 25).

Parameter specification : <n> 0 0 0 0 0 0 0

<n> is treated as an 8-bit binary number,

with the bits being read from most significant to least significant, each 1 indicating a dot and each 0 a space. Thus, for example:

<n> = &FF gives a solid line

<n> = &AA gives a dotted line (dot-space repeated)

<n> = &EE gives a dashed line (dot-dot-dot-space repeated)

The default pattern is <n> = &AA (as in MOS 1.2).

VDU 23 7 Scroll window directly

This sequence allows the current text window (or an arbitrary rectangle on the screen) to be scrolled directly in any direction.

Parameter specification : <m> <d> <z> 0 0 0 0 0

<m> = 0 scroll the current text window
<m> = 1 scroll the entire screen

<d> = 0 scroll right
<d> = 1 scroll left
<d> = 2 scroll down
<d> = 3 scroll up
<d> = 4 scroll in positive X direction
<d> = 5 scroll in negative X direction
<d> = 6 scroll in positive Y direction
<d> = 7 scroll in negative Y direction

<z> = 0 scroll by 1 character cell
<z> = 1 scroll by one character cell vertically, 1 byte horizontally

Notes

1. The cursor is never moved.
2. If <z>=1, horizontal movement depends upon the number of colours in the current mode, i.e.:

   8 pixels in 2-colour modes
   4 pixels in 4-colour modes
   2 pixels in 16-colour modes

VDU 23 8 — Clear block

VDU23 8 causes a block of the current text window to be cleared to the text background colour.

Parameter specification : <t1> <t2> <x1> <y1> <x2> <y2> 0 0

<t1> and <t2> indicate base positions, as defined below. <t1> relates to the start of the block to be cleared, <t2> to the end.

<t1>/<t2>	base position
0	top left of window
1	top of cursor column
2	off top right of window
4	left end of cursor line
5	cursor position
6	off right of cursor line
8	bottom left of window
9	bottom of cursor column
10	off bottom right of window

Note that references to 'left', 'up' etc, are dependent upon the cursor movement control set by VDU 23 16 (see below).

<x1>,<y1> and <x2>,<y2> are displacements from the positions specified by <t1> and <t2> and determine the start and end of the block.

Notes

1. Values of <t1> and <t2> other than those defined above have undefined effects.
2. The result is undefined if the absolute values defining the start and end of the block produce values outside the range −128 to +127.
3. No clearing will take place if the end point of the block lies before the start point.

VDU 23 9 — Set 1st flash time

VDU 23 9 has precisely the same effect as `*FX 9,n`.

Parameter specification : <flash rate> 0 0 0 0 0 0 0

VDU 23 10 — Set 2nd flash time

VDU 23 10 has precisely the same effect as `*FX 10,n`.

Parameter specification : <flash rate> 0 0 0 0 0 0 0

VDU 23 11 — Set default ECF patterns

VDU 23 11 causes the four ECF patterns to be reset to their defaults for the current screen mode.

Parameter specification : 0 0 0 0 0 0 0 0

Notes

The default ECF patterns are as follows, assuming that the default logical-physical map is in effect:

Modes	Pattern	Colour	VDU 23, 2–5 definition (hex)
0 (128)	1	Dark grey	&CC,&00,&CC,&00,&CC,&00,&CC,&00
	2	Grey	&CC,&33,&CC,&33,&CC,&33,&CC,&33
	3	Light grey	&FF,&33,&FF,&33,&FF,&33,&FF,&33
	4	Hatching	&03,&0C,&30,&C0,&03,&0C,&30,&C0
1 (129)	1	Red-orange	&A5,&0F,&A5,&0F,&A5,&0F,&A5,&0F
5 (133)	2	Orange	&A5,&5A,&A5,&5A,&A5,&5A,&A5,&5A
	3	Yellow-orange	&F0,&5A,&F0,&5A,&F0,&5A,&F0,&5A
	4	Cream	&F5,&FA,&F5,&FA,&F5,&FA,&F5,&FA
2 (130)	1	Orange	&0B,&07,&0B,&07,&0B,&07,&0B,&07
	2	Pink	&23,&13,&23,&13,&23,&13,&23,&13
	3	Yellow-green	&0E,&0D,&0E,&0D,&0E,&0D,&0E,&0D
	4	Cream	&1F,&2F,&1F,&2F,&1F,&2F,&1F,&2F
4 (132)	1	Dark grey	&AA,&00,&AA,&00,&AA,&00,&AA,&00
	2	Grey	&AA,&55,&AA,&55,&AA,&55,&AA,&55
	3	Light grey	&FF,&55,&FF,&55,&FF,&55,&FF,&55
	4	Hatching	&11,&22,&44,&88,&11,&22,&44,&88

The mode 0 patterns differ from the mode 4 patterns to prevent problems when the patterns are displayed on televisions.

VDU 23 12–15 Set simple ECF pattern

VDU 23 12–15 allow the ECF patterns to be set in a rather simpler way than that provided by VDU 23 2–5. The drawback is that only a 2x4 pattern of pixels (or double pixels in the case of mode 0) can be set.

VDU 23,12 sets ECF pattern 1
VDU 23,13 sets ECF pattern 2
VDU 23,14 sets ECF pattern 3
VDU 23,15 sets ECF pattern 4

Parameter specification : <a> <b> <c> <d> <e> <f> <g> <h>

The pixels of the top row of the resulting pattern are assigned alternating logical colours <a> and <b>, those of the next row have colours <c> and <d> etc.

Mode 0 uses double pixels to prevent problems when the patterns are displayed on televisions.

VDU 23 16 Cursor movement control

VDU 23 16 allows control of the movement of the cursor after a character has been printed.

Parameter specification : <x> <y> 0 0 0 0 0 0

The normal movement is to the right, with the cursor moving to the start of the next line (scrolling if necessary) if an attempt is made to move it outside the text window.

However, this is all under the control of a byte of flags: VDU 23 16 replaces the current byte by:

((current byte) AND <y>) EOR <x>

The sequence also affects all other sequences which move the text cursor or use coordinates within the text window (e.g. VDU 30).

Notes

The interpretation of the flags is as follows:

bit7 = 0 normal
bit7 = 1 undefined

bit6 = 0 In VDU 5 mode, cursor movements that go beyond the current edge of the window cause special actions (e.g. carriage returns being generated).

bit6 = 1 In VDU 5 mode, cursor movements that go beyond the edge of the window do not cause special actions.

bit5 = 0 Cursor moves in positive X direction after character is printed. If this would result in the cursor moving beyond the edge of the window, the action taken is defined by the settings of bits 6, 4 and 0.

bit5 = 1 Cursor does not move after character is printed.

bit4 = 0 When a cursor movement in the Y direction would result in the cursor being moved beyond the window edge, the window is scrolled if in VDU 4 mode. If in VDU 5 mode, the cursor is moved to the opposite edge of the window.

bit4 = 1 When a cursor movement in the Y direction would result in the cursor being moved beyond the window edge, the cursor is always moved to the opposite edge of the window.

bit3	bit2	bit1		
0	0	0	X direction is right,	Y direction is down
0	0	1	X direction is left,	Y direction is down
0	1	0	X direction is right,	Y direction is up
0	1	1	X direction is left,	Y direction is up
1	0	0	X direction is down,	Y direction is right
1	0	1	X direction is down,	Y direction is left
1	1	0	X direction is up,	Y direction is right
1	1	1	X direction is up,	Y direction is left

bit0 = 0 disables the scroll-protect option. When printing a character in VDU 4 mode would result in the cursor being moved beyond the edge of the window, the cursor is instead moved to the negative X edge of the window and one line in the positive Y direction (what happens if this would result in the cursor being moved beyond the edge of the window is defined by bit 4).

bit0 = 1 enables the scroll protect option. When printing a character in VDU 4 mode would result in the cursor being moved beyond the edge of the window, a "pending carriage return/line feed" is generated. It is actually executed just before the next character is printed, provided it has not been destroyed or executed by another cursor control code (as specified elsewhere).

VDU 23 17–25 Reserved

The commands are currently treated in the same way as the User program calls described below but may be assigned in future.

VDU 23 26 Reserved for Communicator font changes
VDU 23 27 Reserved for Acornsoft sprites
VDU 23 28–31 User program calls

These commands are reserved for use by applications programs, and will result in a call to the unknown PLOT codes vector (locations &226,7). They can be recognised by the following characteristics:

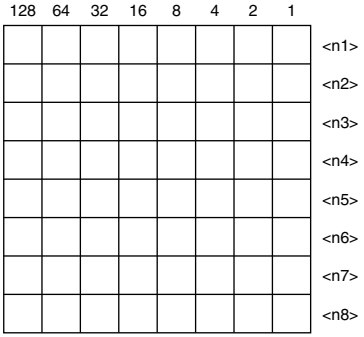
- C=1 on entry to the vector.
- A contains the VDU 23 code (i.e. the first number following 23 in the sequence).
- All of the sequence except the 23 can be found in ascending order starting at VDU variable &1B.

VDU 23 32–255 Define character

VDU commands 23 32–255 redefine the character displayed for each of character codes 32–255.

Parameter specification : <n1> <n2> <n3> <n4> <n5> <n6> <n7> <n8>

<n1>–<n8> are integers representing the eight rows of the character to be redefined (<n1> being the top row, <n2> being the second row etc). Each bit of each value represents one pixel of the corresponding row, with a 1 indicating that the corresponding pixel is to be plotted in the foreground colour and a 0 that it is to be plotted in the background colour (or not at all in the case of VDU 5 mode printing). The most significant bit of the byte corresponds to the leftmost pixel of its row, and the others follow linearly.



VDU 24 (&18) Define graphics window

VDU 24 allows the user to define a window on the screen outside which all graphics will be clipped.

Number of parameters : 8

Parameter specification : <ll> <lh> <bl> <bh> <rl> <rh> <tl> <th>

The four pairs of values define the left, bottom, right and top boundaries of the window respectively (least significant byte first in each case).

Keyboard equivalent : CTRL + X

Notes

1. The four boundary co-ordinates are defined relative to the current graphics origin ((0,0) by default).
2. Any attempt to put the window outside the screen boundaries will be ignored.

VDU 25 (&19) PLOT commands

VDU 25 is a multi-purpose graphics plotting command, taking five parameters. The first parameter defines a particular function, as described below.

These commands are only executed in graphics modes; in non-graphics modes they result in an immediate call to the unknown PLOT codes vector (locations &226,7) – these calls can be recognised by the following:

– C=0 on entry to the vector;
– the computer is not in a graphics mode (this can be recognised by testing VDU variable &61, which contains 0 in non-graphics modes);
– A contains the VDU 25 code (i.e. the first number following 25 in the sequence). The other parameters can be found in ascending order starting at VDU variable &20.

All PLOT commands take the following parameters:

<p> <xl> <xh> <yl> <yh>

where:

<p> is the plot number

<xl> is the least-significant byte of an x co-ordinate relative to the current graphics origin

<xh> is the most-significant byte of the x co-ordinate

<yl> is the least significant byte of a y co-ordinate relative to the current graphics origin

<yh> is the most significant byte of the y co-ordinate

In the descriptions which follow (x,y) is used to denote the point specified by <xl><xh>,<yl><yh>.

When a VDU 25 command is executed, the five parameters are transformed and stored in ascending order in locations starting at a displacement of &1F from the address of the start of the VDU variables. The transformation is:

- If <p> MOD 8 is 0, 1, 2 or 3, the PLOT command is relative to the last graphics point visited and its (external) co-ordinates are added to (x,y)
- If <p> MOD 8 is 4, 5, 6 or 7, the PLOT command is absolute and (x,y) remains unchanged.
- The co-ordinates of the current graphics origin are added to (x,y).
- The resulting co-ordinates are converted to pixel co-ordinates by dividing y by 4 and x by 2 (in mode 0), by 4 (in modes 1 and 4) or 8 (in modes 2 and 5). These co-ordinates are used to specify the 'new point' by all the VDU 25 plots, as specified below.
- A colour and plot mode are also set up up depending on the value of <p> MOD 4:

  <p> MOD 4 = 0 For all but the horizontal line fills and calls that will go to the unknown PLOT codes vector, this causes the plot to terminate rapidly, moving the old graphics cursor to the current graphics cursor, then moving the graphics cursor to the point (x,y).

  For the horizontal line fills and the calls that go to the unknown PLOT codes vector, the colour becomes undefined and the plot mode becomes 5 (i.e. do not change the current value of the pixel when plotting).

<p> MOD 4 = 1 The colour and plot mode are set up to those last defined in a VDU 18 statement whose last parameter was less than 128, i.e. which specified a foreground colour.

<p> MOD 4 = 2 The colour becomes undefined and the plot mode becomes 4 (i.e. invert the current value of the pixel when plotting).

<p> MOD 4 = 3 The colour and plot mode are set up to those last defined in a VDU 18 statement whose last parameter was greater than 127, i.e. which specified a background colour.

Except in the case of rectangular area copy/move, this colour and plot mode is used in all plotting done.

After each of the sequences specified below, except for the horizontal line fills, the old graphics cursor is moved to the current graphics cursor, then the current graphics cursor is moved to the new point.

VDU 25 0–7 Plot solid line (both endpoints included)

These commands cause solid lines to be plotted from the current graphics cursor to the new point.

Such lines will plot exactly the same pixels regardless of the direction of the plot (i.e. a line plotted from A to B will look exactly the same as one plotted from B to A).

BASIC equivalents : VDU 25 4 is equivalent to `MOVE`
VDU 25 5 is equivalent to `DRAW`

VDU 25 8–15 Plot solid line (final point omitted)

These commands cause solid lines to be plotted from the current graphics cursor position to (but not including) the new point.

VDU 25 16–23 Plot dotted line (both endpoints included)

These commands cause a line in the current dot pattern (see VDU 23 6) to be plotted from the current graphics position to the new position. The dot pattern is re-started.

VDU 25 24–31 Plot dotted line (final point omitted)

These commands cause a line in the current dot pattern (see VDU 23 6) to be plotted from the current graphics position to (but not including) the new position). The dot pattern is restarted.

VDU 25 32–39 Plot solid line (initial point omitted)

These commands cause a solid line to be plotted from (but not including) the current graphics position to the new position.

VDU 25 40–47 Plot solid line (both endpoints omitted)

These commands cause a solid line to be plotted from (but not including) the current graphics position to (but not including) the new position.

VDU 25 48–55 Plot dotted line (initial point omitted)

These commands cause a dotted line in the current dot pattern (see VDU 23 6) to be plotted from (but not including) the current graphics position to the new position. The dot pattern is continued from the previous dotted line (or re-started if no dotted lines have been plotted since the last mode change).

VDU 25 56–63 Plot dotted line (both endpoints omitted)

These commands cause a dotted line in the current dot pattern to be plotted from (but not including) the current graphics position to (but not including) the new position. The dot pattern is continued from the previous dotted line (or re-started it if no dotted line has been plotted since the last mode change).

VDU 25 64–71 Plot point

These commands plot a single pixel at the new point.

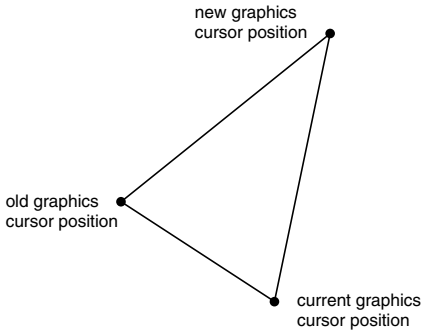
VDU 25 72–79 Horizontal line fill (left and right to non-background)

These commands cause a horizontal line to be plotted in each direction to the left and right of the new point until a pixel which is not of the current background colour (or ECF pattern) is found. The old and current graphics cursors are set to the left and right hand endpoints of this line respectively. Should the new point lie outside the graphics window, or the new point itself be of non-background colour, the error will be indicated by setting the current graphics cursor to the new point and the old graphics cursor to lie on a different horizontal line.

The old and current graphics cursors can be read via OSWORD with A=13.

VDU 25 80–87 Plot triangle

These commands cause a filled triangle to be plotted, with vertices at the old graphics cursor, the current graphics cursor and the new point.



The triangle plotted has the property that it is bounded precisely by the three lines along its edges – i.e. every pixel that would be plotted were these lines to be plotted and every pixel inside the triangle thus formed is included in the triangle plotted, every pixel outside the triangle thus formed is not included in the triangle plotted.

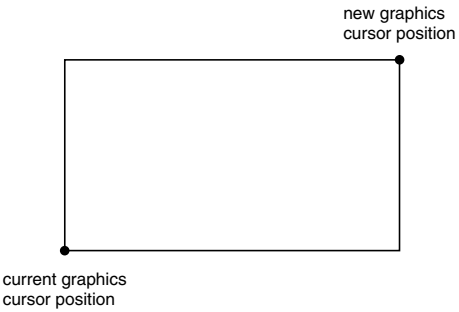
VDU 25 88–95 Horizontal line fill (right to background)

These commands are similar to VDU 25 72–79 above, with the following differences:

- The line is plotted rightwards only from the new point.
- The line is plotted until a pixel of background colour is found.
- If the new point is of non-background colour, there is no error.
- If the new point itself is of background colour, the error is indicated by setting the old graphics cursor to the new point and the current graphics cursor one pixel to its left.

VDU 25 96–103 Plot rectangle

These commands plot a filled, axis-aligned rectangle with opposite corners at the current graphics cursor and the new point.



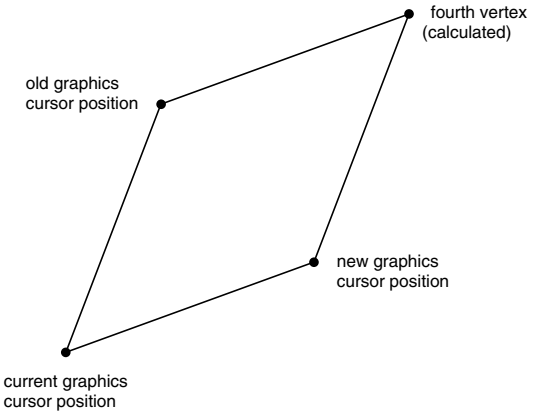
VDU 25 104–111 Horizontal line fill (left and right to foreground)

These commands cause a horizontal line to be plotted in each direction to the left and right of the new point until a pixel which is of the current foreground colour (or ECF pattern) is found. The old and current graphics cursor positions are set to the left and right hand endpoints of this line respectively. Should the new point lie outside the graphics window, or the new point itself be of foreground colour, the error will be indicated by setting the current graphics cursor to the new point and the old graphics cursor to lie on a different horizontal line.

The old and current graphics cursors can be read via OSWORD with A=13.

VDU 25 112–119 Plot parallelogram

These commands plot a filled parallelogram with vertices at the old graphics cursor, the current graphics cursor, the new point and at (new point)−(current graphics cursor)+(old graphics cursor) in cyclic order. The 4th point is calculated in terms of internal pixel coordinates to ensure that the sides are parallel.



The parallelogram is bounded precisely by its four bounding lines in a similar sense to that which applies to triangles (see above).

VDU 25 120–127 Horizontal line fill (right to non-foreground)

These commands are similar to VDU 25 72–79 above, with the following differences:

- The line is plotted rightwards only from the new point.
- The line is plotted until a pixel of non-foreground colour is found.
- If the new point is of non-foreground colour, there is no error.
- If the new point itself is of foreground colour, the error is indicated by setting the old graphics cursor to the new point and the current graphics cursor one pixel to its left.

VDU 25 128–135 Flood fill to non-background

These commands flood fill the screen starting from the new point and continuing until non-background pixels are found. The flooding spreads from pixel to pixel in orthogonal (but not diagonal) directions.

These commands may fail to complete the flood fill if:

– the area to be filled is too complicated;
– the colour being used to fill the area can itself be filled (e.g. plot codes 131 and 135, which fill with background until non-background is found, will fail in this way if the current background plotting mode (specified via VDU 18) is 0);
– an escape condition occurs during the fill.

In each of these cases, the fill will be abandoned at the point at which the problem became apparent.

VDU 25 136–143 Flood fill to foreground

These commands flood fill the screen starting from the new point and continuing until foreground pixels are found. The flooding spreads from pixel to pixel in orthogonal (but not diagonal) directions.

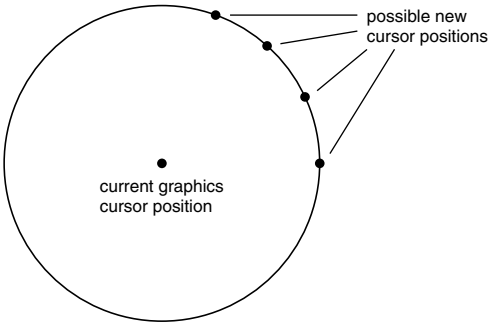
These commands may fail to complete the flood fill if:

– the area to be filled is too complicated;
– the colour being used to fill the area can itself be filled
– an escape condition occurs during the fill.

In each of these cases, the fill will be abandoned at the point at which the problem became apparent.

VDU 25 144–151 Plot circle outline

These sequences plot a circle outline with its centre at the current graphics cursor and the new point on its boundary.



Note

Circles of very high radii cannot be plotted in some modes. The limiting radius depends on the mode, but circles with radius < 16384 (in external coordinates) can always be plotted.

VDU 25 152–159 Plot filled circle

These sequences plot a filled circle with its centre at the current graphics cursor and the new point on its boundary.

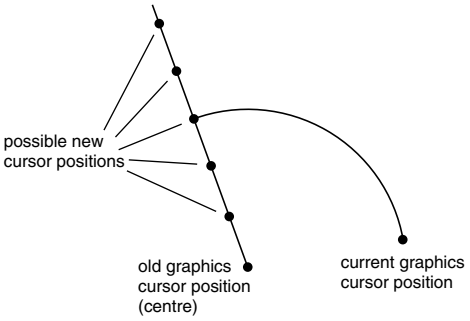
The circle is exactly bounded by its corresponding circle outline, in a similar sense to that in which triangles are exactly bounded by their edges (see above).

Note

Circles of very high radii cannot be plotted in some modes. The limiting radius depends on the mode, but circles with radius < 16384 (in external coordinates) can always be plotted.

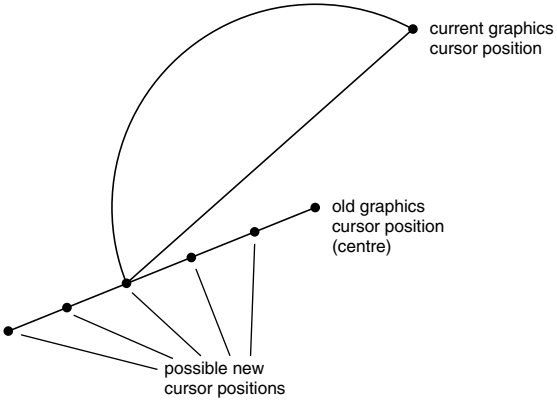
VDU 25 160–167 Plot circular arc

These commands plot a circular arc (only). The centre of the circle is at the old graphics cursor, the first endpoint of the arc is at the current graphics cursor, the second endpoint of the arc is on the circle and in the same direction from the centre of the circle as the new point. The circular arc is taken to be the arc going anticlockwise from the first endpoint to the second.



VDU 25 168–175 Plot filled chord segment

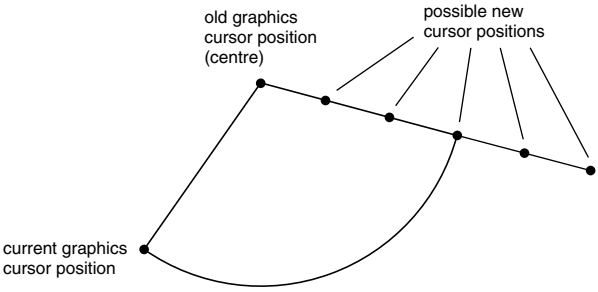
These commands plot a filled area bounded by a circular arc and the chord joining its endpoints. The centre of the circle is at the old graphics cursor, the first endpoint of the arc is at the current graphics cursor, the second endpoint of the arc is on the circle and in the same direction from the centre of the circle as the new point. The circular arc is taken to be the arc going anticlockwise from the first endpoint to the second.



The filled area is exactly bounded by the circle outline and the line which bounds it, in a similar sense to that in which triangles are exactly bounded by their edges.

VDU 25 176–183 Plot filled sector

These commands plot a filled area bounded by a circular arc and the two radii joining its endpoints to the centre of the circle. The centre of the circle is at the old graphics cursor, the first endpoint of the arc is at the current graphics cursor, the second endpoint of the arc is on the circle and in the same direction from the centre of the circle as the new point. The circular arc is taken to be the arc going anticlockwise from the first endpoint to the second.



The filled area is exactly bounded by the circle outline and the lines which bound it, in a similar sense to that in which triangles are exactly bounded by their edges.

VDU 25 184–191 Move/copy rectangle

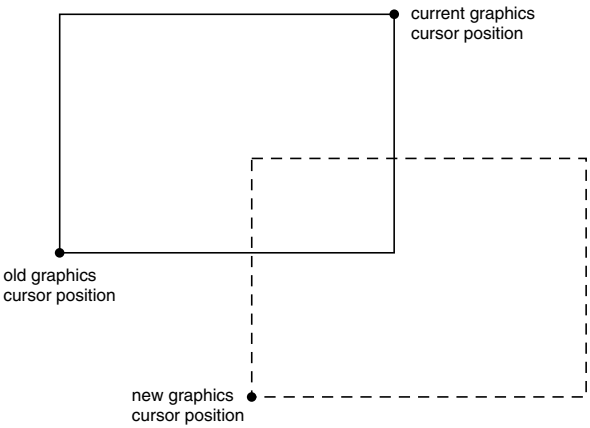
These commands cause the axis aligned rectangle with opposite corners at the old and current graphics positions to be moved or copied so that its bottom left corner is at the new position. The normal interpretation of <p> does not apply in this group of plot codes and the meanings are as follows:

184, 185 } Move rectangle, relative

186, 187 } Copy rectangle, relative

188, 189 } Move rectangle, absolute

190, 191 } Copy rectangle, absolute

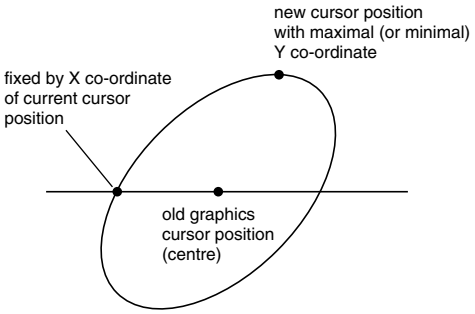


The difference between copying and moving is that moving sets any part of the source rectangle which lies outside the destination rectangle to background, whereas copying leaves such parts of the source rectangle unchanged.

Any part of the source rectangle which lies outside the current graphics window is assumed to contain the current graphics background colour for the purposes of the copy or move.

VDU 25 192–199 Plot ellipse outline

These commands plot an ellipse outline. The centre of the ellipse is at the old graphics cursor. The Y coordinate of the current graphics cursor is ignored, while its X coordinate is that of one of the intercepts of the ellipse outline with the horizontal line passing through the centre. The new point is the highest point of the ellipse (i.e. the point with maximal Y coordinate) if it lies above the centre; otherwise, it is the lowest point of the ellipse.



Note

If an ellipse is too high, numerical inaccuracies may spoil its plotting. This effect starts to be noticeable at about height 2500 (in external coordinates) from the centre of the ellipse to its topmost point.

See general note concerning ellipse plotting under VDU 25 200–207.

VDU 25 200–207 Plot solid ellipse

These commands plot a solid ellipse. The centre of the ellipse is at the old graphics cursor. The Y coordinate of the current graphics cursor is ignored, while its X coordinate is that of one of the intercepts of the ellipse outline with the horizontal line passing through the centre. The new point is the highest point of the ellipse (i.e. the point with maximal Y coordinate) if it lies above the centre; otherwise, it is the lowest point of the ellipse.

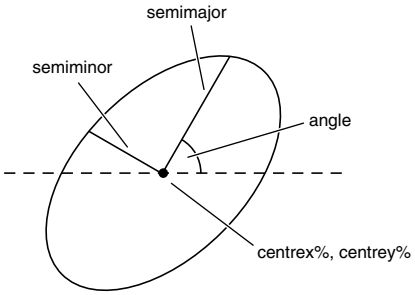
Note

If an ellipse is too high, numerical inaccuracies may spoil its plotting. This effect starts to be noticeable at about height 2500 (in external coordinates) from the centre of the ellipse to its topmost point.

Plotting ellipses

Whilst the algorithm used for plotting ellipses is fast and simple to use for axis aligned ellipses, the required parameters sometimes make it difficult to plot specific non-axis aligned ellipses. In such cases, the ellipse plotting commands may be incorporated into a routine such as that below, which allows an ellipse to be specified in terms of:

– its centre;
– its semi-major and semi-minor axis lengths;
– its rotation from the horizontal axis.




```
1000 DEFPROCellipse(centrex%,centrey%,semimajor,semiminor,angle,
type%)
1010 LOCAL cosang,sinang,maxy,shearx,slicewidth
1020 cosang=COS(RAD(angle))
1030 sinang=SIN(RAD(angle))
1040 maxy=SQR((semiminor*cosang)^2+(semimajor*sinang)^2)
1050 shearx=(semimajor^2-semiminor^2)*cosang*sinang/maxy
1060 slicewidth=semimajor*semiminor/maxy
1070 MOVEcentrex%,centrey%
1080 MOVEcentrex%+slicewidth,centrey%
1090 IFtype%=0THENplotcode%=197ELSEplotcode%=205
1100 PLOTplotcode%,centrex%+shearx,centrey%+maxy
1110 ENDPROC
```

On entry, type%=0 gives an ellipse outline, type%<>0 gives a solid ellipse.

VDU 25 208–231 Reserved

These commands are currently treated in the same way as the following user program calls but may be assigned in future.

VDU 25 232–239 Reserved for Acornsoft sprites

VDU 25 240–255 User program calls

These commands are reserved for use by applications programs, and will result in a call to the unknown PLOT codes vector (locations &226,7). This call can be recognised by the following characteristics:

– C=0 on entry to the vector;
– The computer is in a graphics mode – this can be recognised by testing VDU variable &61. This value is non-zero in graphics modes.
– A contains the VDU 25 code (i.e. the first number following 25 in the sequence). The coordinates can be found in ascending order starting at VDU variable &20.
  Note that these coordinates will have already been transformed as described in the introduction to VDU 25 commands, above.

VDU 26 (&1A) Restore default windows

This command causes the text and graphics windows to be reset back to their default states, i.e. the text window becomes the full screen and is "unset", so that hardware scrolling can be performed, and the graphics window becomes the full screen. VDU 26 also moves the graphics cursor to (0,0), the text cursor to the negative X, negative Y corner of the screen and resets the graphics origin to (0,0).

Number of parameters : 0

Keyboard equivalent : CTRL + Z

VDU 27 (&1B) Null

VDU 27 behaves as VDU 0.

This behaviour is required by the fact that ASCII code 27 is generated by pressing the ESCAPE key.

Number of parameters : 0

Keyboard equivalent : CTRL + [(or ESCAPE)

VDU 28 (&1C) Define text window

This command defines (or redefines) a text window.

Number of parameters : 4

Parameter specification : <lx> <by> <rx> <ty>

The parameters are integers specifying the boundary of the window:

<lx> denotes the leftmost x column
<by> denotes the bottommost y row
<rx> denotes the rightmost x column
<ty> denotes the topmost y row

Keyboard equivalent : CTRL + \

Note

An attempt to define a window which extends outside the screen boundaries for the current mode or which has either <lx> greater than <rx> or <by> less than <ty> will do nothing.

VDU 29 (&1D) Define graphics origin

This command defines the point to be used as the origin for all subsequent graphics output using VDU 24 or VDU 25 commands.

Number of parameters : 4

Parameter specification : <xl> <xh> <yl> <yh>

The two pairs of values specify the (absolute) x and y co-ordinates of the new origin (least significant byte first in each case).

Keyboard equivalent : CTRL +]

VDU 30 (&1E) Home cursor

VDU 30 moves the text cursor to its 'home' position (normally top left, although this may be changed by VDU 23 16).

Number of parameters : 0

Keyboard equivalent : CTRL+^

Notes

1. In VDU 4 mode, any pending carriage return/line feed is destroyed.
2. The text cursor (in VDU 4 mode) or the graphics cursor (in VDU 5 mode) is moved to the negative X, negative Y corner (the "home" position) of the relevant window. Note that the graphics cursor in VDU 5 mode may have an offset of 7 pixels out of the corner along one or both of the axes to allow for the height or width of the character.

VDU 31 (&1F) Tab cursor

VDU 31 moves the text cursor to a specified x and y co-ordinate on the screen.

Number of parameters : 2

Parameter specification : <x> <y>

<x> and <y> are column and row numbers in the range for the current mode.

Keyboard equivalent : CTRL+_

BASIC equivalent : PRINTTAB(<x>,<y>)

Notes

1. In VDU 4 mode, any pending carriage return/line feed is destroyed.
2. In VDU 4 mode, the text cursor is moved to the character position with coordinates <x>,<y>, thinking of the 'home' position (see the previous entry) as having coordinates 0,0, provided this lies inside the text window.
3. If the position lies outside the text window, nothing happens unless the scroll protect option is enabled and the position immediately beyond the positive X edge of the window (i.e. x−1,y lies inside the window). In this case, the text cursor is moved to position <x−1>,<y> and a pending cursor movement to the start of the next line (i.e. to the first character position on the negative X edge of the window) is generated.
4. In VDU 5 mode, the graphics cursor is moved to its 'home' position plus 8*<x> pixels in the positive X direction, plus 8*<y> pixels in the positive Y direction.

VDU 127 (&7F) Backspace and delete

Unless the previous use of VDU 23 16 indicates that no cursor movement is to take place after character printing, the cursor is moved backwards as if by VDU 8 (including the possible clearing of a pending carriage return/line feed). Then the character under the cursor is deleted by overprinting it with a space (in VDU 4 mode) or a solid block in background colour (in VDU 5 mode).

Number of parameters : 0

Keyboard equivalent : DELETE

Note

The characters used during the overprint are selected from the 'hard' (rather than the 'soft') font, so redefining these characters will not change the results.

E.4 The VDU variables

The VDU system stores its variables in 3 places in memory in the I/O processor:

– in a small number of page 0 locations;
– at addresses &300–&37F;
– at addresses &8800–&88FF in the MOS sideways RAM.

It also uses addresses &8400–&87FF in the MOS sideways RAM as workspace (for flood fills).

These variables are generally of interest to the user for a number of reasons:

– to discover the current state of the VDU system;
– to implement a routine to be called from the unknown PLOT codes vector;
– to determine which variables and routines can be used as workspace, etc.

Two OSBYTE calls are provided for interrogating the VDU system although it should be noted that neither returns reliable results if called from an interrupt service routine. Both routines are described in section D.2.2:

OSBYTE 117 (&75) — returns the value of the VDU status byte, returning it in X.

OSBYTE 160 (&A0) — On entry, X contains the number of the required VDU variable (see below).

On exit, X contains the low order byte of the variable and Y contains the high order byte (or the next byte if the variable occupies only a single byte).

In the list of VDU variables below, (p) indicates that the coordinates used are internal pixel coordinates, (e) that they are external coordinates as specified by the user:

Variable	Length	Purpose
&00	2	Graphics window left column. (p)
&02	2	Graphics window bottom row. (p)
&04	2	Graphics window right column. (p)
&06	2	Graphics window top row. (p)
&08	1	Text window left column.
&09	1	Text window bottom row.
&0A	1	Text window right column.
&0B	1	Text window top row.

&0C	2	Graphics origin X coordinate. (e)
&0E	2	Graphics origin Y coordinate. (e)
&10	2	Graphics cursor X coordinate. (e)
&12	2	Graphics cursor Y coordinate. (e)
&14	2	Previous graphics cursor X coordinate. (p)
&16	2	Previous graphics cursor Y coordinate. (p)
&18	1	Text cursor X coordinate (from left of screen) when not cursor editing. When cursor editing, normally the input cursor X coordinate, but the output cursor X coordinate within an "unknown PLOT codes" routine.
&19	1	Text cursor Y coordinate (from top of screen) when not cursor editing. When cursor editing, normally the input cursor Y coordinate, but the output cursor Y coordinate within an "unknown PLOT codes" routine.
&1A	1	Y produced by GADDR (see next section) so that (ZMEMG),Y addresses the correct byte.
&1B–&23		VDU queue: &23 contains last byte of queue, other bytes immediately precede it.
&24	2	Graphics cursor X coordinate. (p)
&26	2	Graphics cursor Y coordinate. (p)
&28–&49		Workspace. But variable &38 must not be used in a Teletext mode (mode 7 or 135).
&4A	2	Address at which the 6845 is to display the text cursor.
&4C	2	Number of bytes in a character row of the text window.
&4E	1	Most significant byte of address of first byte of screen memory.
&4F	1	Number of bytes in a character.
&50	2	Address of byte in top left corner of screen display.
&52	2	Number of bytes in a character row of the whole screen.
&54	1	Most significant byte of number of bytes of screen memory.
&55	1	Current screen mode (in range 0–7, i.e. without regard to 'shadowing').
&56	1	Memory mode: 0 for 20K modes, 1 for 16K modes, 2 for 10K modes, 3 for 8K modes, 4 for 1K modes.
&57	1	Foreground text colour mask.
&58	1	Background text colour mask.
&59	1	0 if plotting graphics foreground, 8 if plotting graphics background.
&5A	1	Current graphics plot mode (usually set to one of following two reduced mod 16).

&5B	1	Current graphics foreground plot mode (as set by VDU 18).
&5C	1	Current graphics background plot mode (as set by VDU 18).
&5D	2	Address of routine to process current VDU sequence.
&5F	1	Value for 6845 register 10 to revert to on leaving cursor editing.
&60	1	(Number of logical colours)−1 (0 if Teletext).
&61	1	(Number of pixels/byte)−1 (0 if not graphics).
&62	1	Mask for leftmost pixel in a byte.
&63	1	Mask for rightmost pixel in a byte.
&64	1	Output cursor X coordinate when cursor editing, but input cursor X coordinate when cursor editing and inside an 'unknown PLOT codes' routine.
&65	1	Output cursor Y coordinate when cursor editing, but input cursor Y coordinate when cursor editing and inside an 'unknown PLOT codes' routine.
&66	1	Cursor control flags (as set by VDU 23 16).
&67	1	Dot pattern (as set by VDU 23 6).
&68	1	Current state of dot pattern.
&69	1	0 if colour being plotted is solid, non-zero if colour is an ECF
&6A	1	0 if graphics foreground colour is solid, non-zero if foreground colour is an ECF
&6B	1	0 if graphics background colour is solid, non-zero if background colour is an ECF
&6C	1	Top bit set when cursor is in "column 81".
&6D	1	Current graphics foreground colour (as set by VDU 18).
&6E	1	Current graphics background colour (as set by VDU 18)
&6F–&7E		Software copy of the current palette.
&7F	1	Reserved.

If the current code is known to be executing in the I/O processor, VDU variable X may be read directly from location &300+X (and &301+X for the high byte of a 2 byte variable) instead of using OSBYTE &A0. This is primarily of interest when programming unknown PLOT codes vector routines – see below.

Using the unknown plot codes vector

The descriptions of VDU 23 and VDU 25 in the previous section give the state of the machine on entering the unknown PLOT codes vector. In addition, a number of other things are known which are likely to aid the programmer considerably:

– The code is known to be executing in the I/O processor, as there is no unknown PLOT codes vector in a second processor. This means that the VDU variables may be read directly from &300+X as specified above, with a considerable saving in speed.

  Variables described as 'reserved' above should not be read or changed by the code, as their usage is not guaranteed to remain constant.

  The variables described as 'workspace' have no guaranteed contents on entry, and may be left in any state on exit from the vector.

  All other variables may be read and changed, but must be left with values which correctly describe the state of the graphics system.

– Similarly, a number of page zero locations may be used directly. These are:

Location	Name	Purpose	Comments
&D0	STATE	VDU status byte.	Do not modify directly.
&D1	ZMASK	Pixel mask.	
&D2	ZORA	Text OR mask.	
&D3	ZEOR	Text EOR mask.	
&D4	ZGORA	Graphics OR mask.	May be used as workspace when not plotting graphics
&D5	ZGEOR	Graphics EOR mask.	
&D6–7	ZMEMG	Graphics pointer.	
&D8–9	ZMEMT	Text pointer.	
&DA–B	ZTEMP	Temporary space.	
&DC–D	ZTEMPB	Temporary space.	
&DE–F	ZTEMPC	Temporary space.	
&E0–1	ZTEMPD	Temporary space.	Used otherwise in BBC micros.

– The MOS sideways RAM and sideways ROM will be paged into the memory map, allowing the following locations to be accessed. Note that these are usually NOT available from outside the unknown PLOT codes vector. Note also that the unknown PLOT codes vector routine should ALWAYS return to its caller straightforwardly via an RTS.

Locations	RAM/ROM	Purpose
&8400–87FF	RAM	VDU workspace.
&8800–07	RAM	ECF pattern 1 definition.
&8808–0F	RAM	ECF pattern 2 definition.
&8810–17	RAM	ECF pattern 3 definition.
&8818–1F	RAM	ECF pattern 4 definition.
&8820–27	RAM	Current foreground ECF pattern or solid colour.
&8828–2F	RAM	Current background ECF pattern or solid colour.
&8830–BF	RAM	VDU workspace.
&88C0–FF	RAM	Reserved for future expansion.
&8900–FF	RAM	Current definitions of characters &20–3F.
&8A00–FF	RAM	Current definitions of characters &40–5F.
&8B00–FF	RAM	Current definitions of characters &60–7F.
&8C00–FF	RAM	Current definitions of characters &80–9F.
&8D00–FF	RAM	Current definitions of characters &A0–BF.
&8E00–FF	RAM	Current definitions of characters &C0–DF.
&8F00–FF	RAM	Current definitions of characters &E0–FF.
&B900–FF	ROM	Default definitions of characters &20–3F.
&BA00–FF	ROM	Default definitions of characters &40–5F.
&BB00–FF	ROM	Default definitions of characters &60–7F (&7F is a solid block like &FF).
&BC00–FF	ROM	Default definitions of characters &80–9F.
&BD00–FF	ROM	Default definitions of characters &A0–BF.
&BE00–FF	ROM	Default definitions of characters &C0–DF.
&BF00–FF	ROM	Default definitions of characters &E0–FF.

The unknown PLOT codes routine should NOT read or write to the screen directly (this will usually fail to read or write correctly in 'shadow' modes) but should use a set of 8 entry points provided in the VDU code for this purpose. Note also that these entry points are ONLY available when the memory map is in the correct state, which is guaranteed on entry to the unknown PLOT codes vector, and usually not true at other times. The 8 entry points are:

&C000 Load a byte of screen memory

`JSR &C000` will perform the equivalent of `LDA (ZMEMG),Y`, taking account of 'shadow' modes.

It takes 17 cycles plus one extra if a page boundary is crossed.

&C003 Store a byte of screen memory

`JSR &C003` will perform the equivalent of `STA (ZMEMG),Y`, taking account of 'shadow' modes.

It takes 18 cycles.

&C006 PLBYTE

`JSR PLBYTE` plots the mask held in ZMASK into the byte pointed to by (ZMEMG),Y, using ZGORA and ZGEOR as colour masks. See GADDR below for an example of its use.

PLBYTE uses ZTEMP as workspace and preserves X, Y, V and C.

&C009 HPLOT

`JSR HPLOT` plots a fast horizontal line in the current graphics colour or ECF and the current graphics mode (all as set by VDU 18) between two specified points. It is the low level primitive used by all the MOS area fill commands.

On entry, two 4 byte areas at &300+X and &300+Y contain the coordinates of the two endpoints, in the standard lowX,highX,lowY,highY order. Should the Y coordinates differ, the Y coordinate of the line plotted is taken from the leftmost of the two points specified.

Only portions of the line inside the graphics window are plotted. Subject to this, both endpoints of the line are plotted.

HPLOT uses ZGORA, ZGEOR, ZMASK, ZMEMG, ZTEMP (but not ZTEMP+1), ZTEMPB, ZTEMPB+1, ZTEMPC and ZTEMPC+1 as workspace. No registers or flags are preserved.

&C00C EIGABS

`JSR EIGABS` converts the 4 byte pair of external coordinates at &300+X (in standard lowX,highX,lowY,highY order) into the corresponding pair of pixel coordinates by offsetting by the graphics origin, then dividing by an appropriate power of 2.

EIGABS uses ZTEMP as workspace, and corrupts all registers and flags.

&C00F WIND

`JSR WIND` windows the 4 byte pair of pixel coordinates (in standard lowX,highX,lowY,highY order) at &300+X, and returns a result in A according to its position with respect to the window:

9	8	10
1	0	2
5	4	6

WIND uses ZTEMP as workspace, preserves X and sets N and Z according to A.

&C012 GADDR

`JSR GADDR` addresses the pixel whose 4 byte pair of pixel coordinates (in standard lowX,highX,lowY,highY order) is at &300+X. GADDR should not be called without first ensuring (typically by means of WIND) that the point concerned does lie within the screen.

GADDR initialises the following variables:

– ZMEMG to the start of the page of memory containing the pixel.

  Y and VDU variable &1A (i.e. location &31A) to contain the offset of the byte containing the pixel within this page – i.e. (ZMEMG),Y points to the byte containing the pixel.

– ZMASK to a mask indicating which bits of this byte constitute the pixel.

– ZGORA and ZGEOR to the correct colour masks for the current graphics plot mode (found in VDU variable &5A) and colour/ECF

– X to Y MOD 7, i.e. the scan line within a character row of the pixel.

Additionally, GADDR uses ZTEMP as workspace and returns A=0, Z=1.

An example of the use of PLBYTE, WIND and GADDR is the following code, which effectively re-implements the VDU 25 64–71 (plot a point) calls. It assumes that the routine addresses have been previously defined and that the graphics plot mode, etc. were set up by the VDU 25 code before the unknown PLOT codes vector was entered:

```
.POINT
 LDX #&20   ;Addresses new point within VDU queue, as
            ;left on entry to the unknown PLOT codes
            ;vector.
 JSR WIND   ;Is the point inside the window?
 BNE END    ;Return if not.
 JSR GADDR  ;Address the point now we know it's on
            ;screen.
 JSR PLBYTE ;And plot the point.
.END
 RTS
```

&C015 IEG

`JSR IEG` takes the internal pixel coordinates of the graphics cursor (in VDU variables &24–&27), converts it back to external coordinates and stores the result in VDU variables &10–&13.

It should be called whenever the graphics code generates a new graphics cursor position (e.g. in the VDU drivers, it is called after a character is printed in VDU 5 mode). Its purpose is to make the two versions of the graphics cursor agree again, and thus prevent errors occurring with relative plots.

IEG uses no page zero locations and corrupts all registers and flags.

Indirecting the unknown PLOT codes vector into a sideways ROM

If the unknown PLOT codes vector is indirected into a sideways ROM via an extended vector (i.e. by indirecting it to &FF39 and putting the address and ROM number in the extended vector space), the MOS routine which does this indirection will change the memory map before entering the sideways ROM. In particular:

– the filing system RAM will occupy addresses &C000–&DFFF;
– the sideways ROM will occupy the entirety of &8000–&BFFF.

This means that the entry points given above are no longer available, and also that data areas in the range &8400–&8FFF are no longer available. This situation can be corrected in two parts:

1. To make the entry points &C000–&C015 available again, your code should be surrounded as follows:

```
LDA &FE34
PHA        ;Preserve current memory map information
LDA #&08
TRB &FE34  ;Make VDU area available
```

(your code)

```
PLA
AND #&08
TSB &FE34  ;Restore memory map
RTS
```

2. To make the data areas in the range &8400–&8FFF available again, your code should be surrounded as follows:

```
LDA &F4
PHA        ;Preserve current ROM number
ORA #&80
STA &F4    ;see note a below
STA &FE30  ;Select sideways RAM
```

(your code)

```
PLA
STA &F4    ;see note a below
STA &FE30  ;Restore original ROM
RTS
```

Notes

a. It is essential to do the `STA &F4` before the `STA &FE30` in both cases, for reasons to do with interrupts;

b. Addresses in the range &8000–&8FFF will now refer to sideways RAM locations, not locations in your sideways ROM. So the entirety of the code above and any subroutines it calls, etc., must lie between addresses &9000 and &BFFF in the sideways ROM.

E.5 The Teletext modes (7 and 135)

The teletext display modes (7 and 135) are handled by the VDU circuitry and VDU driver in quite a different manner to the other display modes. In particular, many of the VDU commands do not work in these modes, their facilities either being altogether absent or provided instead through use of Teletext control characters (ASCII codes in the range 129 – 159).

A list of the principal differences between the teletext and other modes is given below, followed by a list of VDU commands which are still appropriate in modes 7 and 135.

Principal differences

– a different character set, which cannot be redefined (see Appendix B of the Master Series Welcome Guide);

– no logical or physical colour commands – colours selected by characters printed to the screen;

– with minor exceptions (see below), characters are printed by simply writing their ASCII codes to the display memory;

– provision for double height characters;

– no graphics commands;

– no graphics cursor;

– graphics are produced using a block graphics character set.

The exceptions are:

ASCII 35 (hash) which is converted to ASCII 95
ASCII 95 (underline) which is converted to ASCII 96
ASCII 96 (pound) which is converted to ASCII 35

before being written to the screen, to make the teletext character generator display them consistently with the other display modes.

For the teletext specifications adhered to in broadcasting, see “Broadcast Teletext Specification”, available from the BBC or IBA at nominal cost, or the “Teletext System User Guide”, published by Acorn Computers Limited.

VDU commands applicable in the Teletext modes

The following VDU commands behave exactly as described in section E.3:

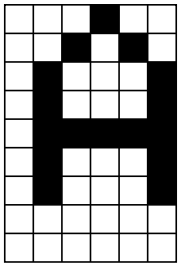
VDU command	Function
0	null
1	send character to printer only
2	enable printer
3	disable printer
6	enable VDU driver
7	produce BELL sound
8	back space
9	forward space
10	next line
11	previous line
12	clear screen
13	carriage return
14	paged mode on
15	paged mode off
21	disable VDU driver
22	select screen mode
23 0	control 6845 directly
23 1	turn cursor off
23 7	scroll window directly
23 8	clear text block
23 16	control cursor movement
23 28–31	user program calls
27	null
30	home cursor
31	tab cursor
127	backspace and delete

The other VDU commands are not obeyed, but parsed and then ignored, with the exception of PLOT commands and unknown VDU 23 commands, which are sent to the unknown plot codes vector.

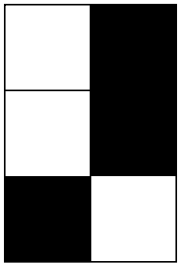
Teletext displayed characters

ASCII codes in the range 32 to 126 and 160 to 255 are displayed as characters, seen either as normal alphanumeric or graphics characters, depending on which, (if any) teletext control codes precede them on the screen line.

Alphanumeric characters are formed from a pattern of 9 by 6 pixels with, for example, the letter A being displayed as:



Graphics characters use a 3 by 2 pattern with each element corresponding to a 3 by 3 square of the pixels used in the formulation of alphanumeric characters. A typical graphics character (ASCII code 186), is:



It is possible to determine the ASCII code for a particular graphics character by adding the 'weights' assigned to each element in the character (see below) and then adding 160.

1	2
4	8
16	64

At the start of each line, the display mode is reset so that characters will be:

– normal height;
– steady;
– white text;
– black background.

If the graphics character set is subsequently selected on any line, it will start as:

– contiguous;
– released.

(see definitions below).

Teletext control codes

The Teletext control codes (ASCII codes in the range 128 to 159) are not VDU commands since they are not executed by the VDU driver, but simply written as normal characters to the display memory. They do however have a similar function to VDU commands, because instead of displaying these codes when it reads one of these codes from the display memory, the teletext display circuitry displays a space (or held graphic character – see below) and interprets it as a command to change display colours, produce graphics rather than text, etc.

Note that in contrast to most VDU commands, the teletext control codes only affect characters to their right on the same line on the screen.

The teletext control codes are listed below, a description of each one is given in the next section. Provided that the base numbers for SHIFT+function key and CTRL+function key have not been changed (using OSBYTE/*FX226 and 227 respectively), certain codes may be obtained directly from the keyboard. Where appropriate, the corresponding keystrokes are given in the table.

control code	effect	default keystrokes
128	null	SHIFT + f0
129	alphanumeric red	SHIFT + f1
130	alphanumeric green	SHIFT + f2
131	alphanumeric yellow	SHIFT + f3
132	alphanumeric blue	SHIFT + f4
133	alphanumeric magenta	SHIFT + f5
134	alphanumeric cyan	SHIFT + f6
135	alphanumeric white	SHIFT + f7
136	flash	SHIFT + f8
137	steady	SHIFT + f9
138	null	
139	null	
140	normal height	
141	double height	
142	null	
143	null	
144	null	CTRL + f0
145	graphic red	CTRL + f1
146	graphic green	CTRL + f2
147	graphic yellow	CTRL + f3
148	graphic blue	CTRL + f4
149	graphic magenta	CTRL + f5
150	graphic cyan	CTRL + f6
151	graphic white	CTRL + f7
152	conceal display	CTRL + f8
153	contiguous graphics	CTRL + f9
154	separated graphics	
155	null	
156	black background	
157	new background	
158	hold graphics	
159	release graphics	

From assembly language programs, Teletext control codes are written to the screen memory using OSWRCH (as for any other character). In the BASIC language, however, the user may either `PRINT` the corresponding character code, eg:

```
PRINTCHR$129"Red on black"
```

or send control codes to the screen using BASIC's `VDU` command, eg:

```
VDU135,157,132:PRINT"Blue on white"
```

The teletext control codes are stored as characters in display memory and are displayed as well as executed by the teletext circuitry. By default they are displayed as a space (a block of the current background colour).

With text output selected, characters are displayed as upper or lower case characters similar to the characters displayed for ASCII 32–126 in the non-teletext modes.

With graphics output selected, some characters are displayed as block graphics (composed of a 2×3 matrix as described above) and some as upper case characters similar to the characters displayed for ASCII 64–90 in the non-teletext modes. Graphics may either be contiguous (so that adjacent blocks meet) or separated (so that adjacent blocks are always separated by a strip in the background colour).

Teletext control code descriptions

Control code 128 **Null**

Control codes 129–135 **Alphanumeric colour specification**

If a code in this range is encountered by the teletext display circuitry, the remainder of the current screen line (or characters up to the next effective control code) will be displayed from the alphanumeric character set in the specified colour.

The current background colour is unchanged.

Control code 136 **Flash**

The flash control code causes the foreground of the remainder of the current screen line (or characters up to a control code 137) to alternate between the current foreground and background colours at a (fixed) rate of approximately 1 Hz.

This is in contrast to the flashing colours available in non-teletext modes which alternate between two colours at a rate which can be specified by the user.

Control code 137 **Steady**

This code cancels the effect of any previous flash code and produces a steady foreground display for the remainder of the current screen line (or characters upto a control code 136.

Control codes 138–139 **Null**

Control code 140 **Normal height**

This control code cancels the effect of any previous control code 141 (see below) for the remainder of the current screen line (or characters upto a further control code 141).

Note, however, that this facility may be used only on the first line of a pair displaying double height characters; use of this code on the second line of the pair will have no effect.

Control code 141 **Double height**

This control code introduces either the first or the second line of a pair displaying double height characters. To be effective, the characters must be displayed in the same position on two successive lines, each preceded by control code 141. If the text on the second line is different, the characters will not match.

Double height output may be switched off on the first line only using control code 140, as described above.

Control codes 142–144 **Null**

Control codes 145–151 **Graphic colour specification**

When a code in this range is encountered by the teletext display circuitry, the remainder of the current screen line (or characters upto the next effective control code) are displayed from the graphic character set in the colour specified.

The current background colour is unchanged.

Control code 152 **Conceal display**

This code causes the remainder of the current screen line (or characters up to the next alphanumeric or graphic colour control code) to be 'concealed' (i.e. displayed as spaces).

Control code 153 **Contiguous graphics**

The occurrence of this code causes graphics characters on the remainder of the current screen line (or characters up to the next effective control code) to be displayed as mutually adjacent blocks (i.e. there is no discernable gap between one graphics character and its neighbours).

Control code 154 **Separated graphics**

In contrast to control code 153, above, this code causes graphics characters on the remainder of the current screen line (or characters up to the next effective control code) to be displayed such that each individual block within each graphics character is separated from all others by a narrow band of the current background colour.

Note that this code does not affect the area of the screen occupied by each graphics character; the effect is achieved by removing the left and lower edges of each block.

Control code 155 **Null**

Control code 156 **Black background**

This code resets the background of the remainder of the current screen line (or characters up to a control code 157 (see below)) to black. The effect starts in the character position occupied by the control code.

Control code 157 **New background**

This control code causes the background for the remainder of the current screen line (or characters up to a control code 156) to be reset to the colour last used to specify the text or graphics foreground colour (i.e. one of control codes 129–135 or 145–151). The effect starts in the character position occupied by the control code.

Control code 157 does not change the foreground colour and it is therefore necessary to select a new foreground colour if subsequent characters on the line are to become visible.

Control code 158 **Hold graphics**

Teletext control codes are normally displayed as a space (i.e. a block in the current background colour). This is normally satisfactory in lines of text output but it means that “holes” will be left in any lines of graphic output.

Control code 158 causes control codes to be displayed as graphics characters; that chosen will be nearest to the left of the control code on that line that has bit 6 set (i.e. ASCII 95–126, 220–254).

If there is no suitable graphics character to ‘hold’ on the current line, or if any of control codes 129–135, 140–141, 145–151 or 159 (see below) intervene, control codes will still be displayed as spaces. The hold graphics facility starts in the character position occupied by the control code, the hold graphics control code itself will also be displayed as the held graphic character if possible.

F Hardware and memory usage

F.1 Introduction

The computer consists of a central processing unit (CPU) with associated memory and various input/output devices for communication with external equipment. These features are summarised below and further information is provided in the sections which follow. A block diagram showing the relationship between the various parts of the system is shown on the next page.

See the Advanced Reference Manual for full, technical specifications.

The CPU and associated memory

All input/output (I/O) computing is performed by a 65C12 CPU, a CMOS device with an instruction set and addressing modes which are a superset of those of the 6502 – see section O in Reference Manual Part 2.

The CPU has a 64K addressing range which is supplemented with a semi-custom integrated circuit (the Memory Controller) to provide up to 320K of memory which may be further increased by the use of suitable plug-in cartridges.

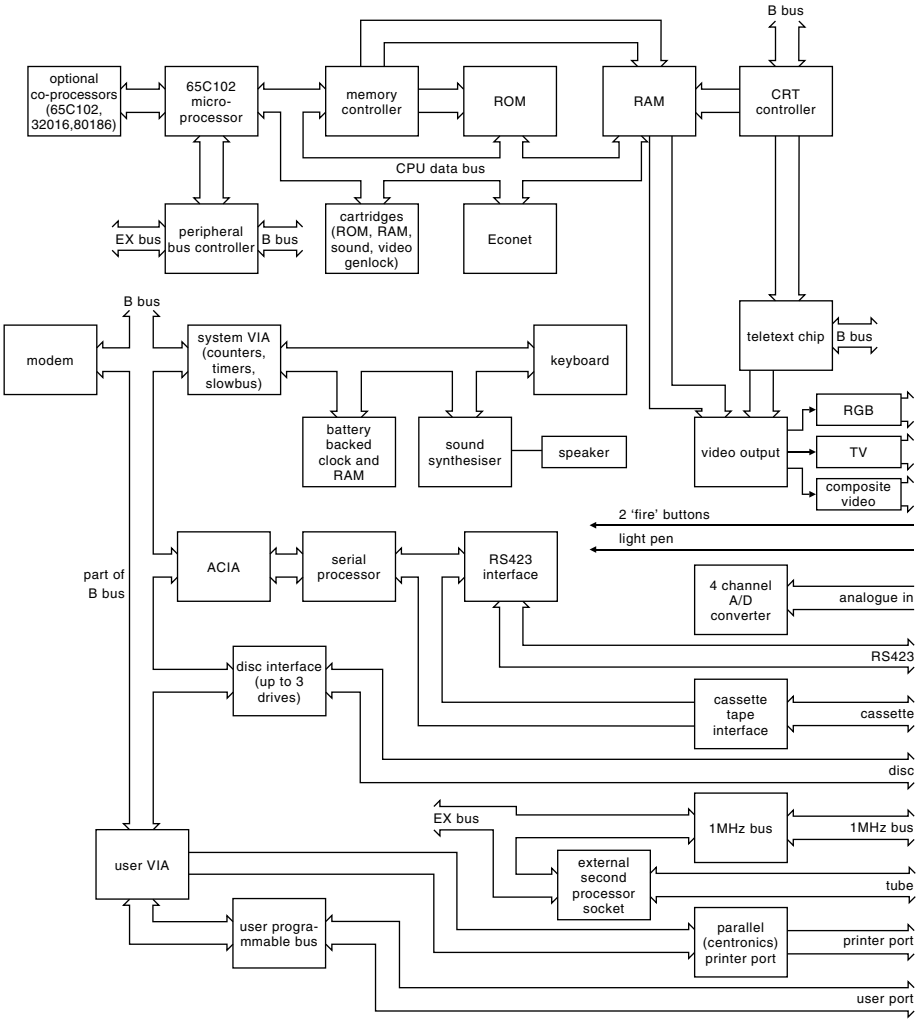
In its standard configuration, the computer has 128K of ROM, provided in the form of a '1 Mbit ROM', and 128K bytes of 120ns Dynamic RAM, provided as:

– 32K RAM addressed in the main memory map;
– 20K Shadow RAM;
– 12K Private RAM;
– 4 x 16K banks of 'sideways' RAM.

Internal I/O

Various I/O devices operate solely within the confines of the system. These devices are:

– a 93 contact keyboard with 2 key rollover;
– a three-channel sound generator with additional noise channel;
– a battery-backed Real-Time Clock and fifty bytes of CMOS RAM;
– various (optional) co-processors.



System block diagram

Co-processors may be either internal (in which case they are plugged directly into connectors mounted on the computer's printed circuit board) or external (in which case they are connected via the TUBE connector). Both types of co-processor consist of an additional CPU with associated memory. They have no I/O capability of their own and depend entirely on the main processor to supply such facilities. When fitted, a co-processor carries out all the computation not associated with I/O operations and, for this reason, is sometimes referred to as the 'language processor'.

External I/O

The following interfaces / subsystems are fitted as standard:

- 6845 CRT controller;
- Four-channel Analogue to Digital Converter;
- Cassette interface;
- Disc Interface and controller;
- Parallel and Serial printer ports;
- 8-bit bi-directional User port;
- RS423 Serial port;
- External 1 MHz bus connector;
- External TUBE connector;
- Audio output to domestic audio equipment.

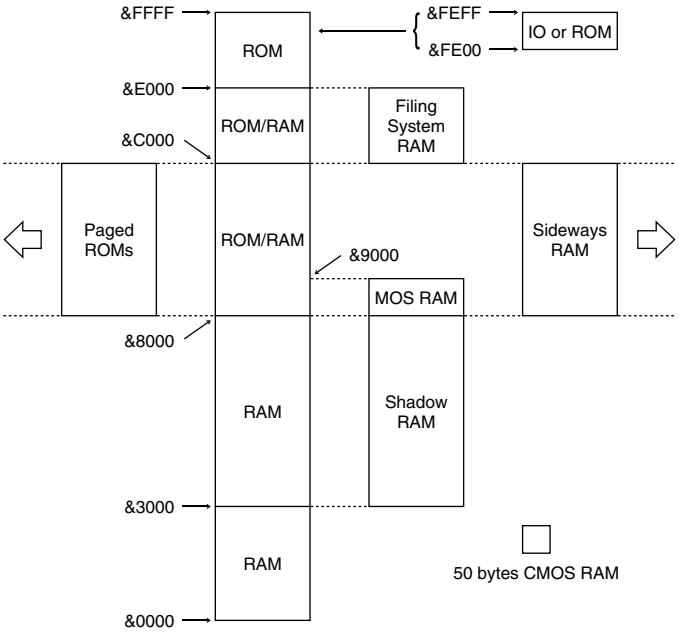
Facilities are also provided for the installation of various optional features, including:

- plug-in Econet network interface module;
- modem installation and connection.

F.2 Memory access control

Operation of the RAM and ROM is controlled by a semi-custom integrated circuit called the Memory Controller. The principal function of this device is to control the memory paging structure.

The memory map is divided into a number of regions, as shown in the diagram below:



The arrangement of the memory map at a given instant is determined by the content of two 'latches':

– the ROM Select Register (**ROMSEL**), located at &FE30.

  The contents of ROMSEL dictate the memory which is accessible in locations &8000 – &BFFF. The bit settings within the register are:

bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
RAM	0	0	0	PM3	PM2	PM1	PM0

RAM (bit 7), when set, causes a 4K segment of private RAM to be paged into the memory map at locations &8000 – &8FFF. This memory segment is used by the MOS during graphics operations and soft key calls and the area is restored to its normal usage once bit 7 is clear.

PM0 – PM3 determine which of the 16, unique, 16K ROM segments is to be paged into locations &8000 – &BFFF.

Seven of the 16 ROM segments (together with a ROM which resides in locations &C000 to &FFFF) make up the 1 Mbit ROM:

ROM No	
15 (&F)	the MOS (incorporating the code for the Cassette Filing System, the ROM Filing System, and the Terminal emulation software)
14 (&E)	VIEW
13 (&D)	The Advanced Disc Filing System
12 (&C)	BASIC
11 (&B)	EDIT
10 (&A)	ViewSheet
9 (&9)	The Disc Filing System

The 1 Mbit ROM is accessed by a separate data bus.

Of the remaining 9 ROM segments, one (ROM number 8) is allocated to the vacant 16K socket on the computer's printed circuit board (IC27); and the others are assumed to be in 4, 32K byte ROMs, where the setting of PM0 in ROMSEL selects between the upper and lower segments. The ROM number allocations are:

ROM No	
7 (&7)	ROM or sideways RAM
6 (&6)	ROM or sideways RAM
5 (&5)	ROM or sideways RAM
4 (&4)	ROM or sideways RAM
3 (&3)	Cartridge ROM/RAM
2 (&2)	Cartridge ROM/RAM
1 (&1)	Cartridge ROM/RAM
0 (&0)	Cartridge ROM/RAM

The computer's printed circuit board contains two (32K) ROM sockets (ICs 41 and 37) corresponding to ROM numbers 4–5 and 6–7. Hardware links (see section F.5) are provided for the purpose of enabling either or both of these sockets if required. Clearly, however, 'permanent' selection of ROM in either or both sockets precludes the use of sideways RAM.

– the Access Control Register (**ACCCON**), located at &FE34.

The contents of ACCCON determine various actions and also dictate the memory which resides in locations &3000 – &7FFF and &C000 – &DFFF. The bit settings in the register are:

bit7	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**
IRR	TST	IFJ	ITU	Y	X	E	D

IRR (bit 7), when set, causes an IRQ to the CPU.

TST (bit 6) must be zero at all times.

IFJ (bit 5), when set causes accesses to locations &FC00 to &FDFF to be directed to the cartridge sockets. When clear, IFJ causes such access to relate to the 1 MHz bus.

ITU (bit 4) relates to accesses to an internal or external co-processor. ITU set enables the CPU to access the internal co-processor; ITU clear enables the CPU to access the external processor.

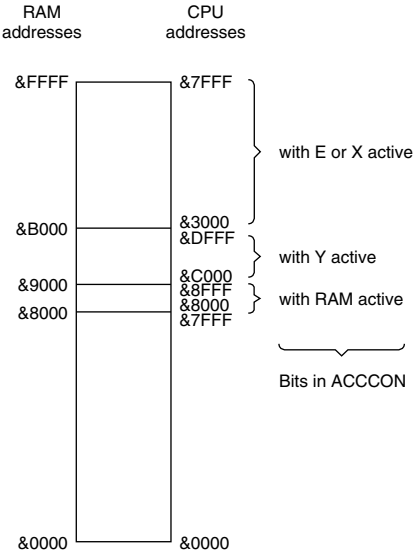
Y (bit 3), when set, causes an 8K segment of private RAM (used by Filing Systems) to be paged into the memory map at locations &C000 – &DFFF, overlaying the VDU driver code which normally resides there. The VDU driver code becomes accessible once Y is clear.

X (bit 2), when set, causes the 20K shadow RAM to be paged into the memory map at locations &3000 – &7FFF. When clear, X restores normal RAM to locations &3000 – &7FFF.

E (bit 1), when set, causes either the main or the shadow RAM to be accessed in locations &3000 – &7FFF depending upon the current operation – RAM in the main map is accessed in all cases except when the address of the current operation code lies in the range &C000 to &DFFF (ie within the MOS VDU driver code), in which case shadow RAM is accessed. If E is clear, all accesses to locations &3000 – &7FFF are directed to the RAM in the main map.

D (bit 0) determines which of main or shadow memory is to be displayed; when set, D causes the CRT controller to display the contents of the 20K shadow RAM; when clear, D causes the controller to display main memory in locations &3000 to &7FFF.

This arrangement is summarised in the diagram below.



F.3 Internal I/O

Various internal components are provided with data from port A of the system 6522, one of the two Versatile Interface Adapters (VIAs). This data is stable until next programmed by the CPU.

The keyboard

The keyboard provides a total of 93 keys; 74 in the main section and 19 in the separate numeric keypad. All keys except BREAK are represented by an entry in a 'keyboard matrix', which is scanned by a semi-custom keyboard encoder. The keyboard matrix is shown below, with C0 – C12 denoting the 13 column lines and R0 – R7 the row lines:

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
									¦				
R7	ESCAPE	f1	f2	f3	f5	f6	f8	f9	\	→	4'	5'	2'
							<	>	?				
R6	TAB	Z	SPACE	V	B	M	,	.	/	COPY	0'	1'	3'
								+	}				
R5	SHIFT LOCK	S	C	G	H	N	L	;	]	DELETE	#'	*'	,'
									*				
R4	CAPS LOCK	A	X	F	Y	J	K	@	:	RETURN	/'	DELETE'	.'
	!	"			&				{				
R3	1	2	D	R	6	U	O	P	[	↑	+'	−'	RETURN'
					'		)		£				
R2	f0	W	E	T	7	I	9	0	_	↓	8'	9'	
		#	$	%		(		=	~				
R1	Q	3	4	5	f4	8	f7	−	^	←	6'	7'	
R0	SHIFT	CTRL											

Entries marked ' denote keys in the numeric keypad.

The BREAK key will reset the CPU and abort any access to the clock/RAM chip. To prevent accidental operation, a mechanical lock is provided in the form of a plastic cam which is rotated through 90 degrees to stop the keytop from being depressed.

During idle (free run) mode, depression of any keys other than SHIFT or CTRL will cause an IRQ to be generated via the system 6522 – these interrupts are controlled by the 6522 control register.

The keyboard column lines (C0 – C12) are continually scanned by incrementing a counter, decoding its outputs and pulling low a column line. Any key depressed will cause the interrupt to be generated. A keyboard enable signal (KBEN) is generated to stop free running mode, at which point the counter contents are loaded by CPU operation to determine on which column the key was pressed. The rows are then individually selected to determine which key was pressed.

The Sound Generator

The sound generator is an SN7694A device which provides three 'sound' channels and one pseudo random 'noise' channel. It is provided with a reference clock of 4MHz from the computer's central timing system. Sallen-Key filters are used to remove all ultra-sonic components of any sound and the output from the filters is mixed with externally applied sound signals and passed to an audio amplifier which drives the internal speaker and the external phono socket.

The output can also be connected (by screened cable) to the optional modem.

Battery backed CMOS clock and RAM

A 146818 Real Time Clock and RAM chip is provided together with a battery backed power supply in the form of a 3.0v Lithium cell mounted in the cartridge 'nest'. The cell supplied will drive the 146818 for a period well in excess of one year without mains power.

As a safety precaution, the 146818 chip select will become invalid if power is removed during an access to the chip. This is done by inverting the chip select with a transistor whose collector resistor is connected to the battery backed supply. As power to the main circuitry fails, the transistor base current reduces and the transistor switches off, deselecting the chip. Even this procedure cannot totally remove the possibility of write access being corrupted.

The computer is also able to accept a nickel cadmium rechargeable battery which may be fitted to the keyboard. Note, however, that whilst the printed circuit board contains the necessary tracking, none of the required support components are fitted and unauthorised modifications to the computer may invalidate the guarantee. This facility is provided mainly for OEM use.

CMOS clock

The clock operates from a 32.768KHz crystal oscillator. A trimming capacitor is provided as is a test point with the buffered clock output. Year, month, day, hour, minute and second information is provided with automatic leap year (but not automatic leap century) correction. An alarm is also included within the chip, but operating system support for this facility is not provided. An optional nIRQ connection can be made to the CPU from the clock chip, enabling the alarm to change program flow. Operation of the clock chip in this manner involves direct manipulation of the chip control signals and should only be attempted by competent programmers. Acorn Computers are not responsible for incorrect programming by the user/software supplier.

CMOS RAM

Fifty bytes of CMOS RAM are available within the chip. Of these bytes:

– 20 are reserved for use by the configuration system;
– 10 are reserved for future use by Acorn;
– 10 are provided for use by commercial third parties;
– 10 are reserved for use by the user.

The allocation of the bytes used by the configuration system is shown below:

Byte offset	**usage**
0	Econet station number
1–2	Econet file server identity
3–4	Econet printer server identity
5	Default Filing System / Language
6–7	ROM frugal bits (set/cleared by `*INSERT`/`*UNPLUG`)
8	EDIT start-up settings
9	reserved for telecommunications applications
10	VDU mode and `*TV` settings
11	ADFS start-up options, keyboard settings and floppy drive parameters
12	Keyboard auto repeat delay
13	Keyboard auto-repeat rate
14	Printer ignore character
15	Default printer type, serial baud rate, ignore status and TUBE select
16	Default serial data format, auto boot option, internal/external TUBE use, BELL amplitude
17–19	reserved for the Advanced Network Filing System (ANFS)

1 MHz Internal I/O

Various devices operate at a 1MHz bus rate although only one internal I/O component works at this speed – the system 6522.

The system 6522 allows several sources to create maskable interrupts. The sources are:

– CRTC vertical synchronisation;
– end of analogue to digital conversion;
– CRTC light pen strobe;
– Keyboard key detect (as described above).

2 MHz Internal I/O

Only one internal I/O component operates at this speed – the internal TUBE used when a co-processor is fitted. Its data bus is connected directly to the CPU data bus.

F.4 External I/O

Printer output

Both parallel (Centronics) and serial printer interfaces are fitted to the computer as standard.

Parallel printer output is handled by port 'A' of the User VIA, timing being maintained by the MOS. Data is buffered by a 74LS144 chip for driving long printer cables – the MOS ensures the accuracy of the data timing on this port.

The parallel printer port (located underneath the computer) takes a female 26-way IDC connector.

Serial printer output is handled by the RS423 serial subsystem (see below), using either the default or a user specified data format and baud rate.

The User port

The User port is used for the connection of peripherals such as:

– Robots;
– Bar code readers;
– Trackerball;
– Mouse units;
– Digitisers;

etc.

and is directly connected to Port 'B' of the User 6522 (VIA). The port provides eight user programmable input/output lines and two handshake lines. The two handshake lines may also be used as a signal generator, for frequency measurement and for generating interrupts.

The 'User port' socket takes a female 20-way IDC connector.

1 MHz External I/O

A variety of devices and subsystems associated with external input/output operations are clocked at 1 MHz, as described below.

Screen Output

Two chips are primarily responsible for providing the screen output:

– the 6845 Cathode Ray Tube Controller (CRTC);
– the Acorn proprietary VIDPROC.

In non-teletext modes, the 6845 generates a linear memory address sequence which increments every 0.5 μs or 1 μs, depending on the video bandwidth selected and video data format. The amount of memory reserved for screen use is also varied, as shown in the table below:

Mode	Format Pixels/Byte	Reserved Memory Bytes	Screen start address	
0	8	20K	&3000	
1	4	20K	&3000	
2	2	20K	&3000	
3	8	16K	&4000	in main memory
4	8	10K	&5800	map
5	4	10K	&5800	
6	8	8K	&6000	
7	Teletext	1K	&7C00	
128	8	20K	&3000	
129	4	20K	&3000	
130	2	20K	&3000	
131	8	20K	&4000	in shadow RAM
132	8	20K	&5800	
133	4	20K	&5800	
134	8	20K	&6000	
135	Teletext	20K	&7C00	

The screen always ends at &7FFF.

Under normal conditions, all modes except 7 and 135 display a bit mapped image of the reserved memory although the 6845 may be re-programmed to display any arbitrary section of memory. If this is done, however, the hardware scrolling will not work correctly, as it assumes that the screen memory is in its usual location.

The selection of video bandwidth, data format and other features is performed by programming the VIDPROC registers. The cursor size and position is also controllable by the VIDPROC. Special measures have been taken to ensure correct cursor operation in the Teletext modes.

The Teletext modes (7 and 135) do not generate a bit mapped display, but a display based on single character cells. The character/graphics ROM within a SAA5050 device generates RGB signals according to the desired character/ graphics information within the reserved memory space. Each byte of memory is therefore just a definition of the character/graphics symbol required.

Only 1K of memory is needed for either of the Teletext modes, although 20K is reserved for it in mode 135. The MOS uses the spare 19K to speed up inter-filing system file transfers but the user may use this memory if no such transfers are to be carried out.

The VIDPROC has to be re-programmed to use the SAA5050 RGB outputs although the 6845 is still used to generate the cursor. As a delay of 2.7 μs will occur after a character is read from RAM, before outputting the appropriate RGB signals, the 6845 has to be programmed accordingly. The start of screen signal is given a 1.5 byte-time offset, and the SAA5050 has a further 1 byte-time offset to restore the correct cursor/data phase.

The VIDPROC has a further adjustment which allows for the cursor to be adjusted to pixel accuracy.

Scrolling may be achieved in any mode by re-programming the 6845 start of screen address to an integral number of video lines further down the memory map than the nominal start of screen. Of course, this causes the linear address generator to attempt to display an end of screen which is outside the reserved video area but this effect is overcome by the provision of a variable address wrap-around during hardware scrolling. In effect, when the address generator would otherwise attempt to access out-of-screen RAM, its addresses are modified to point to the gap between the original start of screen and scrolled start of screen.

Three outputs are provided for displaying video data. These are:

– PAL/NTSC encoded, UHF carrier (via a standard co–axial connector);
– Composite video (via a BNC connector);
– Digital RGB (via a 6-pin DIN plug).

The UHF output is normally PAL encoded (ie suitable for use in the UK); a hardware link option (see section F.5) enables the selection of NTSC encoding.

Analogue to Digital conversion

An NEC UPD7002, 4 channel, 10 bit analogue to digital converter is provided, the sampled input being compared to a 1.8V reference derived from three small signal diodes in series. 10-bit conversions are performed in just under 10ms, with an option of 8-bit conversions (performed in 8ms).

An input voltage on any one of the 4 channels will be digitised when the A/D control register is so instructed. Conversions are in the range 0 to 1.8v and the voltage reference is made available at the connector. Provision for an additional, high-stability reference is made on the main board (see section F.5).

The “End of conversion” pulse causes an IRQ to be generated by the system VIA.

IRQ interrupts (if enabled) are also generated by the following:

– use of one or both joystick ‘fire buttons’;
– a light pen strobe.

The analogue connector is a standard 15-way D-type connector.

Serial I/O

Serial I/O is associated with both the cassette interface and the RS423 subsystem. Much of the circuitry is common to both systems and they are described together below, with any significant differences highlighted.

The devices responsible for providing most of the serial port functions are:

– a 6850 Universal Asynchronous Receiver/Transmitter (UART);
– the Acorn Proprietary SERPROC.

The 6850 UART contains all the receive/transmit and data formatting/error checking facilities necessary for both the RS423 and Cassette systems.

The SERPROC is effectively a multiplexer and baud rate generator for the 6850. It also contains the phase-continuous transmission circuitry for use with the cassette interface.

The RS423 transmit data and CTS lines are buffered by an AM26LS30 chip. This provides a single-ended transmission with slew rate limited output. RS423 receive data and RTS is buffered by a UA9637AC; both buffers are connected with single-ended input configurations.

Cassette data output from the SERPROC is buffered by a single, non-inverting operational amplifier with a simple, single pole filter, AC coupling capacitor and current-limiting output resistor.

The RS423 input/output connector is a 5-pin (‘star’ configuration) DIN plug; the cassette connector is a standard 3-, 5- or 7-pin DIN plug (see Appendix K of the Welcome Guide supplied with the computer).

Disc I/O

A WD1770 series disc controller is fitted as standard. It provides both Frequency Modulated (FM) and Modified Frequency Modulated (MFM) data formats (ie single and double data density modes) for use with 5.25″ floppy and 3.5″ mini-floppy disc drives with a Shugart-compatible interface. Such drives may be powered externally or from the computer’s ‘power out’ socket.

The disc controller performs serial to parallel data conversions (or vice versa), interrupting the CPU via an NMI as bytes come from or go to the disc.

When used in double data density mode, (Modified Frequency Modulated data format), no clock pulses are encoded on the disc and so the WD1770 has a digital phase lock loop to keep track of where data bits should be on the disc. It is important that good quality media and disc drives are used to prevent speed variations affecting reliability.

The disc drive socket takes a female 34-way IDC connector.

Note

The Advanced Disc Filing System keeps a copy of the current directory in memory and will try to load a directory after a reset. If, however, the lack of index pulses from the disc drive indicate that the disc drive is empty, the WD1770 will merely wait for a disc to be inserted. The Advanced Disc Filing System can be instructed not to attempt to load a directory by means of `*CONFIGURE NODIR` (see section C) or by selection using `*FADFS` (see section J).

The external 1 MHz bus

This bus is connected to the Ex-bus on the Peripheral Bus Controller and may be used for connecting a wide variety of expansion devices which require access to the system buses, for example:

- Teletext adapter
- IEEE488 controller
- Test hardware
- Winchester Hard Disc Drives (10 or 30 Mbyte)
- Prestel Adapter
- Music Synthesiser (Acorn Music 500)

During accesses to peripherals on this bus, the CPU has a $\phi 2$ wait state and the Peripheral Bus Controller extends the data cycle to ensure reliable operation with long cables.

The 1 MHz Bus socket takes a female 34-way IDC connector. The cable may be 'daisy-chained' in the attached device to a male 34-way IDC connector so that further devices can be accessed via the bus.

2 MHz External I/O

Two peripheral devices operate at 2 MHz; the external co-processor connection and the Econet connection (used when the optional, plug-in Econet module and the Advanced Network Filing System are fitted).

The external co-processor interface has a buffered data bus via the Peripheral Bus Controller Ex-bus which provides for good data set up and hold times. Together with a limited degree of line matching, this ensures reliable high speed data transfer with unspecified cable lengths. A maximum cable length of one metre is suggested to prevent noise problems.

The interface operates at 2 MHz. This means that if a 1MHz bus peripheral is also connected, then the address and data buses on this connector will appear to perform both 1 and 2 MHz cycles.

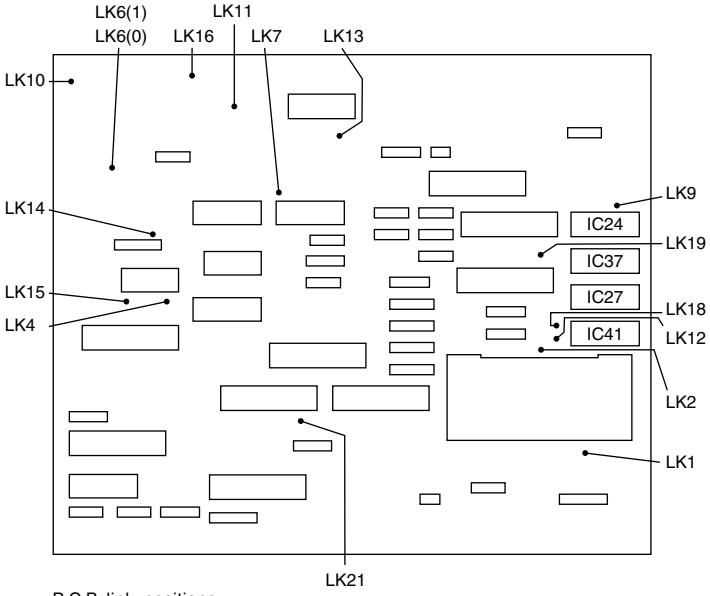
Connection is via a 40-way IDC connector mounted on the underside of the computer.

The Econet connector is driven by the Econet module; a separate printed circuit board containing a 68B54 data link controller chip, line drivers, receivers and 'chatter' disconnect components.

Address, timing, chip select and interrupt signals are taken directly from the CPU data bus by means of a 15-way connector mounted on the main printed circuit board. (Two further address connections are provided for future expansions). A second, 5-way connector provides a one-to-one connection to the external Econet connector which accepts a standard 5-pin DIN plug.

F.5 Hardware links

A number of links are provided on the computer's printed circuit board (see diagram); changes to the state or function of various components may be made by changing, making or breaking the links as described in the remainder of this section.



P.C.B link positions

LK 1 1MHz Bus Audio Input/Output. (Two position link).

In the A position the 1MHz Bus signal is an input to the computer's audio mixer.

In the B position the 1MHz Bus signal is an output from the computer's audio circuit (Minimum load 1K ohm).

This link is a permanent track in the A position. This track must be cut before a wire link is used to make the B position.

LK 2 Cartridge −5V decoupler. (One position link).

In some instances, particular cartridge hardware may need a −5V supply that is decoupled from the main computer −5V load. To do this R9 needs to be fitted and LK2 which is a track on the circuit board should be cut.

LK 3 Not present.

LK 4 Clock chip IRQ. (One position link).

The 6818 clock/RAM chip has a daily alarm function built in. When the alarm is triggered, the CPU is interrupted via its IRQ line. Fitting a shorting link over LK4 connects the CPU IRQ line to the clock line.

This function is not supported by the operating system as this feature may not be present in future versions of the circuit board. Consequently the clock chip must be directly operated by the application software. Note that this will violate ACORN programming recommendations. This link is not fitted as standard.

LK 5 CSYNC polarity. (Two position link).

The polarity of the composite synchronisation signal is determined by this link. It is supplied as a track on the p.c.b. causing negative synchronisation polarity. This track must be broken and a piece of wire used to make the other side of the link for positive synchronisation.

LK 6(0) and LK 6(1) Main Clock select. (Multi-function link).

This group of 4 pins can take either one or two shorting bars as follows:

– Link between A and B – The computer main 16 MHz reference is provided by on-board circuitry. This is normally how computers are shipped.

– Link between B and D – The computer main 16 MHz reference must be provided from pin A17 on either of the cartridge connectors – in this case a clock source MUST be provided or the dynamic memories could be damaged. To prevent this possibility, cartridges providing this facility MUST be fitted in such a way that they are not removable without dismantling the computer. A suitable cartridge locking bar is available from Acorn.

– Link between C and D – The cartridges are clocked by the 8 MHz signal from the computer. This is a synchronous signal with the 2 Mhz (ϕ2) signal, also supplied to the cartridges. Note that the link between A and B must also be fitted.

LK 7 Video polarity. (Two position link).

The polarity of the video RGB signals is determined by this link. It is supplied as a track on the p.c.b. causing true polarity. This track must be broken and a piece of wire used to make the other side of the link for negative polarity.

LK 8 Not present.

LK 9 Megabit ROM select. (Two position link).

As shipped, this link is a track in the WEST position connecting A16 to the megabit MOS ROM. If this track is cut and a wire link used to make the EAST position, a 512Kbit ROM may be plugged into the socket labelled IC24.

LK 10 Channel select. (Two position link).

When used with NTSC V.H.F. televisions, the modulator enables one of two channels to be selected.

Note that the computer as supplied for use in the U.K. is fitted with a U.H.F. modulator and so no channel select and hence LK 10, are provided.

LK 11 Chroma output on composite video. (One position link).

(Issue 2 boards only). This link is not normally fitted. As shipped, the output on the video BNC connector is monochrome. When fitted, this link causes colour information to be encoded onto the video signal.

LK 12 Cartridge Machine Detect / Csync. (Two position link).

Position B – The link position as shipped.

Certain hardware cartridges may need to detect whether they are plugged into a Master Series computer or an Acorn Electron. Master Series computers are shipped with this link in the B position, causing a logic LOW to appear on pin A10 of the cartridges. The Electron has no connection to this pin.

Position A – CSYNC.

This connection to the computer CSYNC line is provided for GENLOCK purposes.

LK 13 A/D converter reference select. (Two position link).

As shipped, this link is a track causing the A/D converter reference voltage input to be 1.8 V. If this track is cut then either of two other voltage references can be used:

- A voltage applied between analogue ground and Vref on the external connector.
- A precision reference can be fitted in the position PR1 if the other side of the link is made with wire.

LK 14 Serial data clock reference. (One position link).

As shipped, this link is a track on the printed circuit board connecting the CHROMA chip 1.23MHz output to the Serial Processor. This link is provided for production purposes and should not be modified.

LK 15 PAL/NTSC select. (Two position link).

As shipped, in the U.K., this link is a track causing the CHROMA chip to encode colour information onto television output in PAL format. If the track is cut and a wire link used to make the other side of the link, then colour information will be encoded in NTSC. In general, televisions within the U.K. can only accept the PAL format.

LK 16 Chrominance information luma trap bypass. (One position link).

This link is not normally fitted. It is provided for those applications where filtering of the luminance information from the chrominance part of the television signal is not required.

LK 17 Not present.

LK 18 Paged ROM/RAM select. (Two position link).

When fitted in the WEST position, this link causes 16 Kbyte of RAM to appear in each of the “sideways” memory “slots” 4 and 5. When fitted in the EAST position, a 32Kbyte ROM occupying slots 4 and 5 may be plugged into the socket labelled IC41.

LK 19 Paged ROM/RAM select. (Two position link).

When fitted in the WEST position, this link causes 16 Kbyte of RAM to appear in each of the “sideways” memory “slots” 6 and 7. When fitted in the EAST position, a 32Kbyte ROM occupying slots 6 and 7 may be plugged into the socket labelled IC37.

LK 20 Not present.

LK 21 Light Pen Strobe to cartridge.

This link is not normally fitted and so position B10 on the cartridges is merely a connection from one to the other. When the link is fitted, the CRTC Light Pen Strobe input is connected to B10. This is to facilitate GENLOCK and an alternative LPSTB connection to the rear analogue connector.

F.6 Memory utilisation

Section F.2 contains a description of the memory map and the means of accessing the various memory segments. This section describes how specific areas of memory are used (mainly by the MOS). Some areas are designated as 'scratch space' and may be used by the user on a temporary basis; others hold essential information which must not be corrupted (although reading is permitted). In general, however, user programs should use only the memory allocated for the purpose and should access all other data using the MOS routines described in section D.

Page 0 (&0000 – &00FF)

&0000–&008F Language workspace.

Programs should not use this area of memory unless they are allowed to do so by the current language. Some languages make part of this page available to other programs (BASIC, for example, reserves locations &70 – &8F for the user) – but this is NOT guaranteed.

&0090–&009F ECONET private workspace.

These locations are used during Econet network operations and should not be used by programs which might possibly be run in a network environment.

&00A0–&00A7 NMI workspace.

These locations are usable only after claiming non-maskable interrupts (NMIs). To use this area, an NMI owner must satisfy two conditions:

– it must have a filing system number allocated to it;
– it must be able to process ROM service calls &B and &C (ie they must either be ROMs (see section F.7) or must intercept calls to OSBYTE 143 (&8F) (see section D.2)).

NMIs should not alter any locations other than their own workspace and locations they have specifically been allowed to alter (e.g. the locations where a file is to be loaded).

&00A8–&00AF MOS scratch space.

This area is corrupted during MOS calls but may be used as user scratch space at other times.

&00B0–&00BF Filing system scratch space.

This area is corrupted during Filing System calls but may be used as user scratch space at other times. Care should be taken to avoid corruption of user data by 'hidden' filing system calls (such as those produced by OSWRCH if a *SPOOL file is in use).

&00C0–&00CF Current filing system workspace.

These locations must be preserved between Filing System calls.

&00D0–&00FF MOS workspace.

These locations may only be used by the MOS and must be preserved at all times.

Page 1 (&0100 – &01FF)

Page 1 contains the 65C12 stack and may also be used as an error message buffer by some paged ROMs. These locations must be preserved at all times.

Page 2 (&0200 – &02FF)

&0200–&0235 Vectors.

These locations are the indirect address entry points for the various MOS and Filing System routines. Programs should not use them except for the express purpose of redirecting specific calls to user-supplied routines.

&0236–&028F Main MOS variables.

Locations in this area contain the main system variables which are accessible using OSBYTE calls 166 (&A6) to 255 (&FF). They should not be used otherwise by programs or written to directly.

&0290–&02FF MOS workspace.

These locations are used during various MOS operations and must be preserved at all times.

Page 3 (&0300 – &03FF)

&0300–&037F VDU variables.

These locations are used to hold information associated with graphics routines (see section E). They must be preserved at all times.

&0380–&03DF Cassette filing system workspace.

These locations must be preserved during operations associated with the Cassette Filing System.

&03E0–&03FF Keyboard input buffer.

These locations must be preserved at all times other than when the user is providing his own keyboard buffering routine(s).

Pages 4 to 7 (&0400 – &07FF)

&0400–&07FF Language workspace.

These locations must be preserved for use by the current language. They should not be used for other purposes unless this is specifically permitted by the current language.

Page 8 (&0800 – &08FF)

&0800-&087F Sound workspace.

These locations must be preserved at all times – their use may produce bizarre sound effects.

&0880–&08BF Printer buffer.

These locations should not be used except when printer output can be guaranteed not to occur.

&08C0–&08FF Sound envelope workspace.

These locations contain the information relating to sound envelopes 1–4 but may be used for other purposes if these envelopes are not used by the current program.

Page 9 (&0900 – &09FF)

&0900–&09BF RS423 output buffer, Cassette output buffer (part) or Sound envelope workspace.

The three uses above are mutually exclusive; when used as sound workspace this area contains information relating to envelopes 5–16. These locations may be used for other purposes if RS423 output, sequential cassette file output and envelopes 5–16 are not used by the current program.

&09C0–&09FF Speech buffer or Cassette output buffer (part).

The two uses are mutually exclusive. These locations may be used for other purposes if sequential cassette file output and speech are not used by the current program.

Page 10 (&0A00 – &0AFF)

&0A00–&0AFF RS423 input buffer or Cassette input buffer.

The two uses are mutually exclusive. These locations may be used for other purposes if sequential cassette file input and RS423 input are not used by the current program.

Pages 11 and 12 (&0B00 – &0CFF)

&0B00–&0CFF Econet workspace.

These locations are used during Econet network operations and should not be used by programs which might possibly be run in a network environment.

Note that this area of the memory map was used for other purposes in previous BBC microcomputer systems.

Page 13 (&0D00 – &0DFF)

Locations in page 13 are either currently assigned for specific purposes or reserved for future use – they should not be used for other purposes.

&0D00–&0D5F NMI routine and workspace.

NMIs must be claimed before this area may be used – see the definition of addresses &00A0–&00A7 above.

&0D60–&0D7F Econet workspace.

These locations are used during Econet network operations and should not be used by programs which might possibly be run in a network environment.

&0D80–&0D91 Currently unallocated.

&0D92–&0D9E Reserved for use by Trackerball.

&0D9F–&0DEF Expanded vector set.

&0DF0–&0DFF Paged ROM workspace.

One byte in this area is assigned to each paged ROM. Each byte is normally used to hold the high byte of the ROM's private workspace address. Some ROMs also use it to indicate that they are not active for some reason – e.g. required hardware not present.

Pages 14 to 127 (&0E00 – &7FFF)

Normally available to the user as language workspace.

The fitment of additional paged ROMs may cause one or more pages at the low end of the region to be allocated as ROM workspace, but only in the event of this requirement not being met by the private RAM – typically by ROMs designed for use with previous BBC microcomputers.

In non-shadow modes, a variable sized area at the high end of the region is used for the screen display (see section F.4).

In shadow modes, the user's language workspace may extend to &7FFF, the screen display being maintained in the shadow RAM.

Pages 128 to 191 (&8000 – &BFFF)

One of 16 paged ROMs resides in this area.

During graphics and soft key calls, the MOS swaps in 4K of the extra 32K of RAM (the part from address &8000 to &8FFF) by setting the high bit of the ROM select register (ROMSEL, see section F.2) thus making paged ROM code in this address range unavailable. Specialist applications (such as graphics extension ROMs) may swap this additional code into memory explicitly if required – see section E.4. Such ROMs must be laid out very carefully, however, in order to avoid attempts to execute ROM code within the overlaid area!

Pages 192 to 251 (&C000 – &FBFF)

This area contains part of the MOS ROM.

In the normal configuration of the machine, pages &C0 to &DF of the MOS are not directly readable, as the filing system RAM is swapped into this area (see section F.2). This part of the MOS contains graphics routines and is swapped in when needed. It is also special in another respect: accesses by instructions in it to data in locations &3000 to &7FFF are automatically mapped into either the main memory or the "shadow" screen memory, depending on the current screen mode. See section F.4 for further information.

Pages 252 to 254 (&FC00 – FEFF)

These locations are reserved for memory mapped I/O and also contain the ROM select register (at &FE30) and the Access control register (at &FE34) which, between them, control the exact state of the memory map at any one time.

Pages &FC (referred to as 'FRED') and &FD (referred to as 'JIM') are for hardware expansion and are designed to be compatible with previous BBC microcomputers. The cartridge interface not only accepts ROM and RAM mapped between &8000 and &BFFF but pages &FC and &FD can also be made to 'appear' in them using OSBYTE 107 (&6B). Unlike the external "1 MHz Bus", accesses to these pages in cartridges are performed at 500ns – twice as fast as the external version.

Two different mappings therefore exist for the allocations within pages &FC and &FD; those associated with the 1 MHz bus are essentially compatible with previous BBC microcomputers, those associated with the cartridge interface are based upon allocations made in the Acorn Electron Plus 1 unit:

	1 MHz bus	**Cartridge interface**
&FC00 – &FC0F	Test hardware	unallocated
&FC10 – &FC13	Teletext	unallocated
&FC14 – &FC17	unallocated	unallocated
&FC18 – &FC1F	reserved	reserved
&FC20 – &FC27	IEEE 488	unallocated
&FC28 – &FC2F	unallocated	reserved
&FC30 – &FC3F	reserved	reserved
&FC40 – &FC47	Winchester disc	unallocated
&FC48 – &FC4F	reserved	unallocated

&FC50 – &FC5F	unallocated	unallocated
&FC60 – &FC6F	unallocated	reserved
&FC70	unallocated	reserved
&FC71	unallocated	reserved
&FC72	unallocated	reserved
&FC73 – &FC7F	unallocated	unallocated
&FC80 – &FC8F	Test hardware	Test hardware
&FC90 – &FC9F	unallocated	reserved
&FCA0 – &FCAF	unallocated	unallocated
&FCB0 – &FCBF	unallocated	reserved
&FCC0 – &FCCF	unallocated	reserved
&FCD0 – &FCDF	unallocated	unallocated
&FCE0 – &FCEF	unallocated	reserved
&FCF0 – &FCFE	unallocated	unallocated
&FCFF	Paged memory register	Paged memory register

Page &FD (with exception of locations &FDFE and &FDFF) is reserved for code. This allows, for example, cartridges to put part of their program(s) into the memory map without paging.

&FDFE and &FDFF are the Test software indirection locations.

Note that RAM cartridges and memory mapped I/O cartridges are not guaranteed to work at the same time. Similarly, 1 MHz devices which generate NMIs (such as Winchester disc drives) should not be accessed whilst cartridge memory mapped I/O is in use. This is because the hard disc interface hardware will have been replaced in the memory map by the cartridges.

Page &FE 'contains' all the internal hardware; some devices are operated at a nominal 1μs bus cycle time, others at 500ns.

In the list below, the base address offset (from &FE00) is given for each device, together with the speed at which it is accessed.

Offset

&00 6845 CRT controller (6845) – 1 MHz

&00	The address register
&01	The data register

&08 6850 Asynchronous Control Interface Adapter (ACIA) – 1 MHz

&08(Write)	Control register
&08(Read)	Status register
&09(Write)	Transmit data register
&09(Read)	Receive data register

&10 The Serial Processor (SERPROC) – 1 MHz

&18 Analogue to Digital Converter (UAPD7002) – 1 MHz

&18(Write)	Data latch and start of conversion
&18(Read)	Status register
&19(Read)	Data high byte
&1A(Read)	Data low byte

&20 The Video Processor (VIDPROC) – 1 MHz

&20(Write)	Video Control Register
&21(Write)	Palette

&24 The Floppy Disc Interface Control Register – 2 MHz

&28 The Floppy Disc Controller (1770) – 1 MHz

&30 The ROM Select Register (ROMSEL) – 2 MHz

&34 The Access Control Register (ACCCON) – 2 MHz

&38 Network NMI disable (INTOFF) – 2 MHz

&3C Network NMI enable (INTON) – 2 MHz

&40 System 6522 (VIA) – 1 MHz

&40(Write)	Output Slow Peripheral Control Bus
&40(Read)	Input Slow Peripheral Control Bus
&41(Write)	Output Slow Peripheral Data Bus
&41(Read)	Input Slow Peripheral Data Bus
&42	Data Direction Slow Peripheral Control Bus
&43	Data Direction Slow Peripheral Data Bus

&44(Write)	Timer 1 low order latch
&44(Read)	Timer 1 low order counter
&45(Read)	Timer 1 high order counter
&46	Timer 1 low order latch
&47(Write)	Timer 1 high order latch
&48(Write)	Timer 2 low order latch
&48(Read)	Timer 2 low order counter
&49	Timer 2 high order counter
&4A	Shift Register
&4B	Auxiliary Control Register
&4C	Peripheral Control Register
&4D	Interrupt Flag Register
&4E	Interrupt Enable Register
&4F(Write)	Output Slow Peripheral Data Bus (No handshake)
&4F(Read)	Input Slow Peripheral Data Bus (No handshake)

&60 User 6522 (VIA) – 1 MHz

&60(Write)	Output User Port
&60(Read)	Input User Port
&61(Write)	Output Printer Data
&61(Read)	Input register A {Data not valid}
&62	Data Direction User Port
&63	Data Direction Printer Data
&64(Write)	Timer 1 low order latch
&64(Read)	Timer 1 low order counter
&65	Timer 1 high order counter
&66	Timer 1 low order latch
&67(Write)	Timer 1 high order latch
&68(Write)	Timer 2 low order latch
&68(Read)	Timer 2 low order counter
&69	Timer 2 high order counter
&6A	Shift Register
&6B	Auxiliary Control Register
&6C	Peripheral Control Register
&6D	Interrupt Flag Register
&6E	Interrupt Enable Register
&6F(Write)	Output Printer Data (No handshake)
&6F(Read)	Input register A {Data Not Valid}

&80 Third party hardware (typically a modem) – 1 MHz

&A0 Network Interface – 2 MHz

&E0 TUBE – 2 MHz

Page 255 (&FF00 – &FFFF)

These locations are used exclusively by the MOS.

Shadow and Private RAM

The 32K of additional RAM does not occupy a single contiguous block of addresses. Instead, it is allocated as follows:

&3000–&7FFF **Shadow screen memory.**

In shadow modes, this 20K RAM segment is used for the screen display – any unused part (i.e. locations lower than the start of screen address for the current screen mode) is free for other purposes. In non-shadow modes, this area can be accessed by suitable manipulation of the Access Control register (ACCCON). Note that certain MOS operations (such as `*MOVE`) do this as a matter of course.

&8000–&8FFF **Various functions.**

This 4K RAM segment is paged into the memory map for a variety of functions associated with MOS and VDU driver functions:

&8000–&83FF Soft key expansions buffer.

This area is used by the MOS during soft key calls and expansions.

&8400–&87FF VDU workspace.

This area is used by VDU driver routines which need large amounts of workspace (currently the 'flood fill' operations only).

&8800–&88FF VDU variables and workspace.

This area is used by the VDU driver, partly for holding information relating to the four Extended Colour Fill (ECF) patterns and partly as workspace.

&8900–&8FFF Character definitions.

This area contains the current display definitions for character codes 32 to 255 (&20 – &FF). This area is sometimes referred to as the 'soft font'.

&C000–&DBFF **Paged ROM workspace.**

This area can be claimed by ROMs (via service calls) using a similar method to that by which they may claim workspace above OSHWM. Note that only filing systems should use absolute (static) workspace, either in this area or above OSHWM – any ROM may have 'private' workspace.

&DC00–&DCFF **MOS CLI buffer.**

Corrupted by all '*' commands.

&DD00–&DEFF **Transient utility workspace.**

This area is used by `*MOVE` and is otherwise available to the user for implementing non-standard '*' commands.

&DF00–&DFFF **MOS private workspace.**

F.7 Paged ROMs

Paged ROMs must always contain two types of information:

– header information;
– service code.

They may also contain other code depending upon their purpose (ie a language, Filing System, Utility ROM etc).

The header information is used by the MOS to identify and address the functions provided by the ROM; the service code is used to allow the ROM to claim resources and to respond to requests from the MOS or other ROMs. Any remaining code is executed if and when the ROM is selected as the current language / Filing System – a ROM may provide a combination of language / Filing System / Utility functions provided that the appropriate header and service code are provided.

The following sections describe the format of a paged ROM header and provide general information relating to ROM service calls. Those wishing to produce their own paged ROM code (for any purpose) should consult the Advanced Reference Manual.

ROM header information

Paged ROMs always occupy locations &8000 – &BFFF in the main memory map and the ROM header information is located at the start of this area:

Location	Content
&8000	&4C (ie a 65C12 JMP instruction)
&8001	<LSB of ROM language entry address>
&8002	<MSB of ROM language entry address>
&8003	&4C (ie a 65C12 JMP instruction)
&8004	<LSB of ROM service call entry address>
&8005	<MSB of ROM service call entry address>
&8006	ROM type flag (see below)
&8007	<offset to start of copyright message>
&8008	<version number in binary>
&8009	title string (displayed when selected as a language and by `*ROMS`)

The title string is terminated by a byte containing &00 and may be followed by an (optional) version number string terminated by a further byte containing &00. The (optional) version number string is followed by a copyright message which must start with the sequence:

&28 (ASCII '(') (pointed to by location &8007)
&43 (ASCII 'C')
&29 (ASCII ')')

The copyright message is terminated by a byte containing &00. If necessary, it may be followed by four bytes containing the relocation address (LSB first) if the code is designed to be executed by a co-processor.

The ROM type flag (&8006) provides information about the ROM:

bits	indication when set
7	ROM has a service entry (all ROMs should have this bit set)
6	ROM is a language
5	ROM contains code assembled to run in a co-processor
4	not used
3–0	code contained in the ROM: 0000 : 65C12 (or 6502) BASIC 0001 : Reserved 0010 : 65C12 (or 6502) code other than BASIC 0011 : 68000 code 1000 : Z80 code 1001 : 32016 code 1010 : Reserved 1011 : 80186 code 1100 : 80286 code 1101 : Reserved

The language entry address is called by the MOS after initialising the ROM as a language (or copying it across to the co-processor). These bytes should contain &0000 (and bit 6 of the ROM type flag should be clear) if the ROM does not contain a language.

The service call entry address is called whenever a component of the system wishes a ROM to provide a service, as described below:

Service code

ROMs may be requested to perform a variety of services, each identified by a service call number. When required, service calls are passed to the available paged ROMs in descending order of ROM socket number (starting with the 'MOS ROM' in socket 15 (&F). In each case, the code pointed to by the ROM service code entry address will be entered for each request, with:

A=<service call number> (see below)
X=<ROM number>
Y=<optional parameter>

The ROM's service code should check the content of the accumulator and ignore call numbers to which the ROM does not respond (ie it should return immediately using an RTS). A, X and Y should normally be preserved.

For those call numbers to which the ROM is able to respond, the ROM service code should carry out the necessary actions. On completion of the service the ROM may set A to zero (ie 'claim' the call, which will prevent the call from being passed to subsequent ROMs) before returning. Note that some services are intended to be passed to all paged ROMs – calls of this nature should not be claimed. Unless the ROM wishes to carry out some major function (such as selecting itself as the current language using OSBYTE 142), it should return using an RTS.

If the service returns a parameter, it will normally do so in Y.

A list of the various paged ROM service call numbers and their functions are given below.

&00 No operation.

&01 Claim filing static workspace allocation.

Static workspace is allocated on hard resets only, it starts at &E00 and may only be used by the current Filing System. Current top of workspace is passed in Y and should be incremented by any filing system requiring more (Y contains the first unused page number).

This call must not be claimed.

&02 Offer dynamic workspace allocation.

Dynamic workspace is available to any sideways ROM and is allocated on hard resets. The Y register contains the page number of the next available page, and a paged ROM requiring dynamic workspace should increment Y by the number of pages it requires before returning. A ROM may store the page number of the start of its dynamic workspace at location &DF0+<ROM number>. The ROM number may be found in location &F4.

This call must not be claimed.

&03 Offer bootstrap.

This call offers a filing system the opportunity to start as the current filing system. If the Y register is zero on entry then the filing system should auto-boot if it starts (typically by running or execing a file !BOOT). A filing system which starts up should claim this call.

&04 Offer command.

This call provides an opportunity for any sideways ROM to execute a command not recognised by the MOS CLI. If it does so it should claim the call.

The command may be found at (&F2),Y terminated by carriage return. Note that filing systems should not attempt to run files or execute any of their specific commands at this call.

&05 Unknown interrupt.

An interrupt has occurred which is not recognised by the MOS. A sideways ROM may handle this interrupt, in which case it should clear the cause of the interrupt and claim the call. If the call is not claimed then the interrupt will eventually be passed to the user.

&06 Offer BRK.

A BRK instruction has occurred and sideways ROMs may take any appropriate action that they want.

This call should not be claimed.

&07 Offer unknown OSBYTE.

An OSBYTE call has occurred which is not handled by the MOS. A sideways ROM may claim this call and take whatever action is appropriate. If the call is not claimed then the OSBYTE call will be ignored, unless generated by the CLI, in which case a `Bad command` error will be reported. A filing system which is not currently selected should either ignore any OSBYTEs from this source or select itself before executing them.

&08 Offer unknown OSWORD.

An OSWORD call has occurred which is not handled by the MOS. A sideways ROM may claim this call and take whatever action is appropriate. If the call is not claimed then the OSWORD call will be ignored. A filing system which is not currently selected should either ignore any OSWORDs from this source or select itself before executing them.

This is the recommended route for communication between application programs and sideways ROMs where speed is required.

&09 *HELP facility.

The `*HELP` command has been detected by the CLI, and sideways ROMs have an opportunity to give helpful information where appropriate.

Every paged ROM should respond with its name and a list of help commands it accepts if no parameter is supplied. If `*HELP` is followed by a list of arguments then a sideways ROM should only respond if it recognises one of the arguments.

This call should not be claimed.

&0A Claim static filing system workspace.

This call may only be issued by a filing system when starting up, in which case the previous filing system will save away any data of importance in its private workspace. Note that some filing systems (such as the Cassette Filing System) may not need to claim static workspace.

&0B Release NMI.

Return ownership of the NMI resource plus workspace to the previous owner. Because NMIs are unmaskable, only one task may be allowed to produce NMIs at once. The Y register should contain the previous owner when this call is made.

&0C Claim NMI.

Force current owner of the NMI resource and workspace to relinquish ownership. There is no way to reject this call. This call should only ever be made from foreground tasks. On return, the Y register will contain the previous owner.

&0D Initialise ROM Filing System

&0E Get byte from ROM Filing System.

&0F Changed filing system indirections.

This call warns any other paged ROMs that may wish to intercept filing system indirections that they have now been changed as a new filing system is starting.

&10 Shut *EXEC and *SPOOL/*SPOOLON files.

This call is made by the MOS when an OSBYTE 119 call is found.

&11 Unused

This call was used in previous BBC microcomputer systems for telling paged ROMs about font implosions and explosions.

&12 Start filing system.

The parameter passed in the Y register is the filing system number, NOT the ROM number of the ROM containing that filing system. Some ROMs may contain more than one filing system, e.g. DNFS.

&15 Offer polling interrupt to paged ROMs.

If a paged ROM centisecond polling interrupt has been requested (using OSBYTE 22) then this call will be made on every timer interrupt. The Y register will contain the number of outstanding requests for this interrupt. A sideways ROM may decrement Y by the number of requests it has made. Any sideways ROM decrementing Y to zero may claim the call.

&18 Reserved.

&21 Offer static workspace in filing system RAM.

This call is analogous to call &01 but offers RAM in the filing system workspace area starting at &C000. The top of this workspace is &DC00.

This call should not be claimed.

&22 Offer dynamic workspace in filing system RAM.

This call is analogous to call &02 but offers RAM in the filing system workspace area. The top of this workspace is &DC00.

This call should not be claimed.

&23 Inform top of static workspace in filing system RAM.

Many ROMs can use more static workspace than they need to claim for extra buffers. This call informs all ROMs of the full size of the filing system static workspace in the filing system workspace area.

This call should not be claimed.

&24 Count dynamic workspace in filing system RAM.

The Y register contains the current bottom of dynamic allocation and should be decremented by the number of pages required by each ROM.

This call should not be claimed.

&25 Request filing system information.

This call requests the filing system name (up to eight characters), its low and high handle limits and its filing system number. The

information should be returned in an 11 byte block starting at (&F2),Y. The Y register should be returned incremented by the number of bytes added to the table (a multiple of eleven).

Filing systems may have more than one name. The filing system name should be space padded to eight characters. The next two bytes are the low and high handle limits, and the last byte is the filing system number.

This call should not be claimed.

&26 Close all open files.

This call is a request for all filing systems with open files to select themselves and close all their open files. It is used by `*SHUT`.

This call should not be claimed.

&27 Inform reset

A reset has occurred and paged ROMs should initialise themselves.

This call is required because the MOS does not offer workspace on soft break and should not be claimed.

&28 Unknown configuration command.

A paged ROM owning a portion of the CMOS RAM may use this command to update its own configuration information.

&29 Unknown status command.

A paged ROM owning a portion of the CMOS RAM may use this command to display its own configuration information.

&2A A language change is about to occur.

This is to enable languages like Terminal to remove their interception of buffering functions etc.

&2B Reserved.

&FE Tube post-initialisation.

&FF Initialise tube software.

G Filing Systems

G.1 Introduction to Filing Systems

A Filing System is a collection of routines (similar to those making up the MOS) provided for the purpose of organising and accessing data held on external storage media such as cassette tapes, floppy discs, hard (Winchester) discs and (to a more limited extent) on ROM cartridges.

Four Filing Systems are fitted in the computer as standard and a summary of their characteristics is given in the table below. The facilities provided by each Filing System are described in detail in sections H, I and J. Users who have not previously used a disc system are advised to read section 5 of the Welcome Guide supplied with the computer.

Filing System Name	**Storage Medium**	**Storage device**	**Speed**	**Capacity**
Cassette Filing System (CFS)	cassette tape	cassette recorder	slow	high
ROM Filing System (RFS)	ROM/EPROM cartridge	built-in	medium	low
Disc Filing System (DFS)	floppy disc	floppy disc drive	fast	medium
Advanced Disc Filing System (ADFS)	floppy/disc	floppy disc drive	fast	high
	hard disc	winchester disc drive	v fast	v high

Other Filing Systems (particularly the Advanced Network Filing System (ANFS)) may be fitted to the computer if required and interested users should contact either Acorn or their nearest Acorn approved dealer for further information.

The default Filing System

The default Filing System is that which is to be selected when the computer is switched on or subjected to a hard break and it is specified by the use of the command:

`*CONFIGURE FILE` <rom id>

followed by a hard break. This has the effect of storing the ROM number of the default filing system in the least significant four bits of location 5 in the CMOS RAM – the most significant four bits of this location hold the ROM number of the default language.

In the standard machine configuration, the <rom id> parameters for the various Filing Systems are:

	<rom id>
CFS	15 (&F)
DFS	9 (&9)
ADFS	13 (&D)

Note that it is not possible to select the ROM Filing System as the default.

Once selected, the default Filing System becomes the 'current' Filing System and it remains operative until another Filing System is selected using one of the methods described below.

Changing the current Filing System

The current filing system may be changed by one of the following methods:

– from the keyboard

Some Filing Systems may be selected directly from the keyboard by depressing a designated <selection key> in conjunction with **BREAK** or **CTRL**+**BREAK**:

	<selection key>
CFS	**SPACE**
DFS	`D`
ADFS	`A` or `F` (see section J)

– from the command mode of the current language

Any Filing System may be selected using the appropriate <filing system selection command>:

<filing system selection command>

CFS : `*TAPE`, `*TAPE12` or `*TAPE3` (see section H)
RFS : `*ROM`
DFS : `*DISC`
ADFS : `*ADFS` or `*FADFS` (see section J)

The above commands may also be incorporated in BASIC programs and in programs written in languages with access to the OSCLI routine.

– from assembly language

using OSBYTE 143 with X=18 (&12) and Y=<filing system number> as defined below:

<filing system number>

CFS : 1 (1200 baud) or 2 (300 baud) (see section H)
RFS : 3
DFS : 4
ADFS : 8

Using the current Filing System

The facilities provided by the current Filing System are available to the user in a number of ways:

– via Filing System commands

Filing System *<text> commands may be used in BASIC programs or in programs written in languages with access to the OSCLI routine. They may also be issued directly from the command mode of most languages (but see also below).

Filing System commands fall into two categories:

– 'common' commands (ie those commands which are generally applicable to all Filing Systems), which are recognised by the MOS;
– 'intrinsic' commands (ie those commands which are specific to a particular Filing System), which are not recognised by the MOS and are offered in turn to the paged ROMs and ultimately to the current Filing System as 'unknown' MOS commands.

Common commands are described later in this section; see the section covering the appropriate Filing System for a description of the various intrinsic commands.

– via language implementations of Filing System commands

such as the LOAD and SAVE commands provided in BASIC, VIEW and ViewSheet, f2 (Load File), SHIFT+f2 (Insert File) and f3 (Save File) in the Editor.

See the appropriate section in either the VIEW or ViewSheet Guides or Reference Manual Part 2 for descriptions of these commands.

– via MOS routines

normally in assembly language programs.

A summary of the available routines is given in section G.6; descriptions are given in the the chapters devoted to the various filing systems.

Temporary use of a different Filing System

Although only one Filing System may be current at any one time, it is possible to specify temporary use of a different Filing System by including the name of the Filing System in commands. The <filing system name> parameters for the various Filing System names are:

		<filing system name>
CFS	:	-TAPE- (or -CFS-)
RFS	:	-ROM-
DFS	:	-DISC-
ADFS	:	-ADFS-

Where this technique is used, the format of a *<text> command becomes either:

*<filing system name><text> [parameter(s)]

or:

*<text> <filing system name>[parameter(s)]

The first format is mandatory if the <text> part of the command is an intrinsic command for the named Filing System.

Note that there are a very small number of instances where sensible execution of the command in question requires the named temporary Filing System to be made the current Filing System – see OSBYTE 109 (&6D).

A Filing System other than the current Filing System may also be specified as the 'Library' Filing System as described in the next section.

G.2 How files appear to the user

Files are merely sequences of bytes held on a storage medium – their content has no significance until they are processed in some way by another component of the system. Thus, for example, a file might be:

– a sequence of characters created using the Editor;
– a BASIC program or a program written in some other language;
– a machine code program;
– a data file, created by running a program;
– a VIEW document;
– a ViewSheet spreadsheet;

etc.

Filing Systems make no distinction between types of file such as those listed above and the user is at perfect liberty to, say, load a VIEW document with BASIC as the current language – it is only when the file has been loaded that the content of the file will be found to be inconsistent with the format required by the language. Similarly, whilst it is perfectly possible to load a machine code program with the Editor as the current language, the result will be far from useful.

In general, problems of this nature can be avoided by sensible use of the facilities provided by Filing Systems, viz:

– choice of appropriate file names;
– use (where possible) of the Filing System structure;

as described in the remainder of this section.

File names

Files stored on an external medium are normally identified by means of a <file name> which can often be used to indicate the purpose or type of a file. The maximum length of a file name and the characters which may be used are Filing System dependent:

CFS : 10 characters
RFS : 10 characters
DFS : 7 characters
ADFS : 10 characters

The characters used to make up a <file name> may be taken from the set of numbers and letters (no distinction being made between upper and lower case) – other characters should be used with caution as they may have a special

significance, depending upon the current Filing System:

- is used to introduce temporary Filing System names (see section G.1);

. is used to separate directory specifications under the Disc and Advanced Disc Filing Systems (see below);

: is used to introduce a drive specification under the Disc and Advanced Disc Filing Systems (see below);

* is used as a 'wildcard' (matching a group of characters) under the Disc and Advanced Disc Filing Systems (see section J);

is used as a 'wildcard' (matching a single character) under the Disc and Advanced Disc Filing Systems (see section J);

$ is the name of the root directory under the Advanced Disc Filing System (see section J);

& is an alternative to $ under the Advanced Disc Filing System (this is provided for compatibility with the Advanced Network Filing System (ANFS));

@ is used to denote the current directory under the Advanced Disc Filing System (see section J);

^ is used to denote the 'parent' directory under the Advanced Disc Filing System (see section J).

File names used under either the Disc or Advanced Disc Filing Systems must be unique within a given directory, as described below.

Note. In the case of the cassette and ROM Filing Systems, in which access to files is in strictly serial sequence, it is sometimes possible to specify a 'null' file name – in which case the next file on the medium will be accessed. There is also no real need (other than those dictated by common sense) for file names to be unique on these media.

Directories

Under the Disc and Advanced Disc Filing Systems, it is possible to group files together into units referred to as directories and this facility is often used to keep files of a particular type together. The means of identifying a directory is Filing System dependent:

<directory name>

DFS : 1 character (any except <space>)
ADFS : 10 characters (as for file names)

In both cases, the first access to the Filing System selects directory **$** (sometimes referred to as the 'root directory'), which becomes the 'current' directory.

Files in the current directory may be accessed without reference to the directory name; access to files in another directory may be made either by making the desired directory the current directory (see section J) or by prefixing the file name by an appropriate <directory specification>:

<directory specification>

DFS : <directory name>**.**
ADFS : <pathname>

where <pathname> = [**$.**][<directory name>**.**] ... (ie a sequence of directory names starting from either the root directory or the current directory, each separated by a **.** character).

Drives

Both the Disc and Advanced Disc Filing Systems allow files to be distributed across all the available disc surfaces. Under the Disc Filing System, each disc surface is denoted by a separate drive number; under the Advanced Disc Filing System, both surfaces of double sided discs are treated as a single entity.

Filing System	**Disc type**	**max number of drives**	**<drive number>**
DFS	floppy	4	0, 1, 2 and 3
ADFS	floppy only	2	0 and 1 (or 4 and 5)
ADFS	hard only	1	0
ADFS	floppy/hard	1 hard	0
		2 floppy	4 and 5

Note that possible future expansions will allow the Advanced Disc Filing System to access up to four hard disc drives (numbered 0, 1, 2 and 3 and/or upto four (double-sided) floppy disc drives (numbered 0, 1, 2 and 3 (or 4, 5, 6 and 7)).

Note also that under the Advanced Disc Filing System (only), drive 'numbers' 0 to 7 may be specified as A – H respectively.

In the case of both the Disc and the Advanced Disc Filing System, the first access selects the lowest numbered drive for the type(s) of disc available, which becomes the 'current' drive.

Files located on the current drive may be accessed without specifying a drive number; access to files on another drive (if available) may be made either by making the desired drive the current drive (see section J) or by prefixing the file name (and optional directory specification) with a <drive specification>, which is defined as:

<drive specification> = **:**<drive number>**.**

Libraries

Machine code routines stored on disc may be executed by means of a command of the form:

*****<file name>

(provided that <file name> is neither a MOS command name nor a Filing System command name).

Both the Disc and the Advanced Disc Filing Systems provide means of specifying a 'library directory' (library) to hold such routines, specified by means of a drive number and directory using a command as described in section J.7.

When a *****<file name> command is encountered, the Filing System first checks for the existence of <file name> in the current directory, loading and running the named code if it is found. If <file name> is not an entry in the current directory, the procedure is repeated for the library directory (if set). The message:

```
Not found
```

is reported if <file name> cannot be located in either directory.

In addition to the library associated with the current Filing System, it is possible to specify a 'Library Filing System' using the the command:

*****<filing system name>`LIBFS`

Where this facility is in use, the existence of the <file name> in a *****<file name> command is checked in the following sequence:

1. The current directory of the current Filing System.
2. The library associated with the current Filing System.
3. The current directory of the Library Filing System.
4. The library associated with the Library Filing System.

G.3 How files appear to the Filing System

In order to be able to access a specified file, each Filing System maintains information relating to all the files on the medium associated with the Filing System.

Under the Cassette and ROM Filing Systems, files are divided into segments referred to as blocks, which are between 0 and 256 bytes in length. Each block contains (amongst other information):

- header information;
  - file name;
  - load address;
  - execution address;
  - block number;
  - length of data in block;
  - access information;
- data bytes.

For a full specification, see sections H.5 and I.5.

Under the Disc and Advanced Disc Filing Systems, file information is held in 'catalogues' in reserved areas of the medium in use, that for each file (amongst other information) being:

- file name;
- load address;
- execution address;
- length of file in bytes;
- access information;
- start sector on disc.

For a full specification, see sections J.5 and J.10.

G.4 Summary of common Filing System commands

The Filing System commands listed below are those which are recognised by the MOS and which can, therefore, be considered to be common to all Filing Systems. Note, however, that some common commands are either inappropriate or meaningless with either the Cassette or ROM Filing Systems operative. An indication of the implementation of commands under these Filing Systems is given under the heading 'Exceptions' in the table below.

Descriptions of each command can be found in section G.5.

Command	Description	Exception(s)
*APPEND	Append lines of keyboard input to a file	CFS, RFS
*BUILD	Send lines of keyboard input to a file	RFS
*CAT	Display the file catalogue	
*CLOSE	Close all sequential files	
*CREATE	Create a file of given size	CFS, RFS
*DELETE	Delete a file name from the file catalogue	CFS, RFS
*DUMP	Display a file as a hex. and ASCII dump	
*EX	Display file catalogue information	
*EXEC	Take subsequent lines of input from a file	
*INFO	Display information for specific file(s)	CFS, RFS
*LIST	Display a file (with line numbers)	
*LIBFS	Select the Library Filing System	
*LOAD	Load file into memory	
*MOVE	Move file	
*OPT	Set Filing System options	
*PRINT	Display file in pure ASCII	
*REMOVE	Delete a file name suppressing 'Not found' errors	CFS, RFS
*ROM	Select the ROM Filing System	
*RUN	Load and execute a machine code program	
*SAVE	Save a block of memory as a file	RFS
*SHUT	Close all files on all active Filing Systems	
*SPOOL	Send all VDU output to a file	RFS
*SPOOLON	Append all VDU output to a file	CFS, RFS
*TAPE	Select the Cassette Filing System	
*TYPE	Display a file (without line numbers)	

G.5 Common Filing System command descriptions

This section contains a description of the 'common' Filing System commands. They are presented in alphabetical order and, where appropriate, the minimum permissible abbreviation is shown for each command.

In all cases, the command applies to the current Filing System unless either the text of the command name or the parameter to the command is prefixed with a Filing System name, in which case it applies to the named, temporary Filing System.

The syntax definitions for some commands contain references to <object>, which is a unique file reference dependent upon the current (or temporary) Filing system:

		<object>
CFS	:	<file name>
RFS	:	<file name>
DFS	:	[<drive specification>][<directory specification>]<file name>
ADFS	:	[<drive specification>][<pathname>]<file name>

Definitions of the constituent terms are as given in section G.2 – any new definitions are provided in the corresponding description.

Note that the Advanced Disc Filing System provides a number of 'shorthand' facilities which enable long pathnames and file names to be abbreviated – see section J.7 for further information.

A number of error messages may be produced by incorrect or invalid use of either commands or parameters. Full lists of the error messages associated with the various Filing Systems can be found in sections H.4, I.4, J.4 and J.8.

As with MOS commands, the case of the characters in the command is not significant.

*APPEND *AP.

Syntax

`*APPEND` <object>

Purpose

To append subsequent lines of keyboard input to an existing file.

Description

*APPEND opens the existing, named file and sets the file pointer to the end of the file, ready to accept subsequent lines of keyboard input. Input is terminated by the occurrence of an escape condition which causes the file to be closed.

See *BUILD below.

*BUILD *BU.

Syntax

*BUILD <object>

Purpose

To create a new file containing subsequent lines of keyboard input.

Description

*BUILD opens a new file with the specified name; all subsequent lines of keyboard input are directed to the file, input being terminated by the occurrence of an escape condition, which causes the file to be closed.

In the case of the Disc and Advanced Disc Filing Systems, an existing file of the same name is lost, unless protected (see section J).

*BUILD sets the execution address for the file to −1 (&FFFFFFFF), which causes any attempt to *RUN the file to be treated as a *EXEC command (see below).

*CAT *.

Syntax

CFS : *CAT
DFS : *CAT [:][<drive number>]
ADFS : *CAT [<drive specification>][<pathname>]

Purpose

To display the names in a file catalogue.

Description

Under the Cassette and ROM Filing Systems, *CAT displays a list of the file names held on the corresponding medium – any parameter(s) are ignored. See also *OPT below.

Under the Disc Filing system, a catalogue of the current (or specified drive) is displayed; under the Advanced Disc Filing System, the display shows the content of the current (or specified) drive/directory.

*CLOSE *CL.

Syntax

`*CLOSE`

Purpose

To close all open sequential files on the current Filing System.

Description

`*CLOSE` ensures that any open sequential files associated with the current Filing System are closed properly (ie that any unwritten data remaining in RAM is written to the file). It is equivalent in effect to BASICs `CLOSE#0` statement.

*CREATE *CR.

Syntax

`*CREATE` <object> <size in bytes>

Purpose

To create an empty file of a specified length.

Description

`*CREATE` provides a simple means of reserving space for files on disc. <size in bytes> is taken as a hexadecimal number indicating the number of bytes to be reserved.

Note that the action is to reserve space only – no initial data is written to the file.

*DELETE *DE.

Syntax

`*DELETE` <object>

Purpose

To delete a file name from the current file catalogue.

Description

Provided that the specified object exists, its name is deleted from the associated file catalogue – an error message is displayed if the object cannot be found (see `*REMOVE` below).

Note that the effect of *DELETE is merely to delete the name of the file – the sectors of the disc occupied by the content of the file are left unchanged and become part of the 'free space map' for the disc in question. All or part of such areas may be subsequently used to store files of an appropriate size.

See also the descriptions of *COMPACT, *FREE and *MAP in section J.

*DUMP *D.

Syntax

*DUMP <object> [<start point>] [<offset>]

Purpose

To display a dump of a file.

Description

*DUMP displays a hexadecimal and ASCII dump of the named file, with eight bytes per line, preceded by a value indicating the displacement of the leftmost byte from the start of the file:

<displacement> <8 hexadecimal bytes> <8 ASCII characters>

'non-printable' characters are shown as a .

The two optional parameters specify the point at which the dump is to commence and the initial value of the displacement shown on the display. <start point> defaults to zero; if omitted, <offset> defaults to <start point>.

*EX

Syntax

CFS : *EX
RFS : *EX
DFS : *EX [<drive specification>][<directory specification>]
ADFS : *EX [<drive specification>][<pathname>]

Purpose

To display file catalogue information.

Description

Under the Cassette and ROM Filing Systems information is displayed for all files on the medium – any parameter(s) are ignored.

Under the Disc and Advanced Disc Filing Systems, information is displayed for all files in the current (or specified) directory.

The information displayed for each file is:

<file name> <attributes> <load address> <execution address> <length> <start sector>

All values are shown in hexadecimal; see section J for a description of file attributes.

*EXEC *E.

Syntax

`*EXEC` <object>

Purpose

To take subsequent input from a file.

Description

`*EXEC` reads all the data in a named file, byte by byte, as if it was being typed from the keyboard. (Such files are normally created using the `*BUILD` command, as described above).

*INFO *I.

Syntax

`*INFO` <object>

Purpose

To display Filing System information for file or group of files.

Description

`*INFO` displays the same Filing System information as `*EX` (see above) but for either a single file or a group of files.

A single file is specified by explicit reference; groups of files are specified using the two 'wildcard' characters # and *:

# denotes any single character;
* denotes any number of characters.

*LIST *LI.

Syntax

*LIST <object>

Purpose

To display a file with line numbers.

Description

*LIST displays the content of the named file in GSREAD format, in which:

– ASCII codes 32 to 126 are sent directly to the screen as the corresponding character in the current mode;
– all other ASCII codes (except 13) are displayed as control code expansions.

Each line (ie sequence of ASCII codes terminated by a carriage return) is preceded with a line number (starting at line 1).

*LIBFS *LIB.

Syntax

*<filing system name>LIBFS

Purpose

To define the Library Filing System.

Description

The *LIBFS command sets the Library Filing System to that named, as described in section G.2.

*LOAD *L.

Syntax

*LOAD <object> [<load address>]

Purpose

To load a file into memory.

Description

If no <load address> parameter is given, *LOAD loads the specified file into memory starting at the load address held for the file by the named (or temporary) Filing System. If <load address> is specified, it is taken to be a hexadecimal value and it overrides the load address held by the Filing System.

*MOVE

Syntax

*MOVE [<filing system name>]<object 1> [<filing system name>]<object 2>

Purpose

To move (copy) files, either between drives and/or directories or between Filing Systems.

Description

*MOVE transfers a copy of <object 1> from either the current or specified Filing System to a new file (referenced by <object 2>) on either the current or specified Filing System.

When used without <filing system name> parameters, *MOVE provides a convenient means of producing a copy of a file.

*OPT *O.

Syntax

*OPT <option number> [<option value>]

Purpose

To set Filing System options.

Description

The permissible values for <option number> are Filing System dependent:

		<option number>
CFS	:	0, 1, 2 and 3
RFS	:	0 and 1
DFS	:	0, 1 and 4
ADFS	:	0, 1 and 4

*OPT0 (or merely *OPT) restores the default settings for all options associated with the current Filing System.

*OPT1 controls the display of file information during load and save operations.

<option values> 0 and 1 are applicable to all Filing Systems and specify that file information is to be either suppressed or displayed.

<option value> 2 applies only to the Cassette Filing System and specifies that the load address and execution addresses are to be displayed once a file has been loaded.

`*OPT2` controls the action to be taken in the event of an error.

<option value>	0 specifies that errors are to be ignored
<option value>	1 prompts for a re-try
<option value>	2 aborts the operation with an error message.

`*OPT3` controls the length of the interblock gap during single-byte output to cassette files.

<option value> may lie between 1 and 127 and specifies the length of the interblock gap in tenths of a second. A value greater than 127 will reset the length to its default of 2.5 seconds.

`*OPT4` controls the auto-start option for disc systems.

<option value>	0 disables the auto-start facility
<option value>	1 causes the `!BOOT` file to be `*LOAD`ed
<option value>	2 causes the `!BOOT` file to be `*RUN`
<option value>	3 causes the `!BOOT` file to be `*EXEC`ed

*PRINT *P.

Syntax

`*PRINT` <object>

Purpose

To display a file in pure ASCII.

Description

`*PRINT` displays the content of the named file in pure ASCII, ie it sends each byte to the VDU driver regardless of whether it is a printable character or a control code. It may therefore produce unpredictable results with files containing anything other than simple text.

*REMOVE *RE.

Syntax

`*REMOVE` <object>

Purpose

To delete a file from the current file catalogue.

Description

`*REMOVE` is identical in effect to `*DELETE` with the exception that the `Not found` error is suppressed if the file does not exist.

*ROM

Syntax

`*ROM`

Purpose

To select the ROM Filing System.

*RUN */

Syntax

`*RUN` <object> [<optional parameters>]

Purpose

To load and execute a machine code program.

Description

`*RUN` loads the named file into memory and starts execution (using the load and execution addresses stored by the Filing System). <optional parameters> are ignored by the Cassette and ROM Filing Systems but may be accessed by the program under other Filing Systems.

Under the Disc and Advanced Disc Filing Systems, `*RUN` <object> may be abbreviated to `*`<object> unless the object name is either a MOS or Filing System command or a command recognised by a paged ROM (eg `*WORD`).

If the execution address for the specified object is −1 (&FFFFFFFF), `*RUN` is equivalent to `*EXEC`.

*SAVE *S.

Syntax

*SAVE <object> <start addr> <finish addr> [<exec addr> [<reload addr>]]
or

*SAVE <object> <start addr>+<length> [<exec addr> [<reload addr>]]

Purpose

To save a block of memory as a file.

Description

*SAVE takes a copy of a designated area of memory and writes it to the named file on the current (or temporary) Filing System.

<start addr>	is the address of the first byte to be saved
<finish addr>	is the address of the byte after the last byte to be saved
<length>	is the number of bytes to be saved
<exec addr>	is the execution address to be stored with the file (this value is assumed to be the same as <start addr> if omitted)
<reload addr>	is the reload address, (which allows the load address stored with the file to be different to the actual start address used when saving the file and which is assumed to be the same as <start addr> if omitted)

All address parameters to *SAVE are treated as hexadecimal values.

*SHUT

Syntax

*SHUT

Purpose

To close all currently open sequential files on all Filing Systems.

Description

*SHUT is similar in effect to *CLOSE although it affects all, rather than just the current, Filing Systems.

*SPOOL *SP.

Syntax

*SPOOL <object>

Purpose

To send all VDU output to a file.

Description

*SPOOL used with a parameter opens a new file with the specified name and causes all subsequent VDU output to be directed to it. The spooling process is terminated with the execution of either another *SPOOL <object> command (in which case the previous file is closed and spooling continues with the new file) or a *SPOOL command with no parameter (which closes the current file).

*SPOOLON

Syntax

*SPOOLON <object>

Purpose

To append all VDU output to an existing file.

Description

As for *SPOOL but with an existing file, to which all subsequent VDU output is appended.

The file may be closed using either a *SPOOL or *SPOOLON command without a parameter.

*TAPE *T.

Syntax

*TAPE *or*
*TAPE3 *or*
*TAPE12

Purpose

To select the Cassette Filing System.

Description

*TAPE or its variants select the Cassette Filing System – see section H for further information.

*TYPE *TY.

Syntax

`*TYPE` <object>

Purpose

To display a file.

Description

`*TYPE` displays the content of the named file in GSREAD format (as for `*LIST`). In this case, however, no line numbers are generated.

G.6 Using Filing Systems from assembly language

A number of routines are provided in order to make Filing Systems accessible to assembly language programmers and, as with the MOS routines described in section D, each has a name and a call address, as shown below.

Routine name	**address**	**function**
OSFIND	&FFCE	Open or close a file for byte access
OSGBPB	&FFD1	Load or save a number of bytes
OSBPUT	&FFD4	Write one byte to an open file
OSBGET	&FFD7	Read one byte from an open file
OSARGS	&FFDA	Load or save data about an open file
OSFILE	&FFDD	Load or save a complete file
OSWORD	&FFF1	Load or save a number of sectors (and other functions)

Whilst the principle of the operation of these routines is applicable to all Filing Systems, use normally depends upon the current Filing System and descriptions are therefore left to the corresponding sections.

G.7 Utilisation of sideways RAM

In its standard configuration, the computer is provided with four 16K 'banks' (64K) of sideways RAM. By default, each bank of sideways RAM is designated as an area of memory into which suitable ROM images may be loaded although it is possible to specify use of each bank for other purposes, as described below.

The default settings are reset only on power-up.

Addressing modes

The sideways RAM may be addressed in one of two ways:

– absolute addressing

in which case each page of sideways RAM is referenced by means of a <rom id> and which may be paged into the main memory map at locations &8000 – &BFFF;

– pseudo addressing

in which case one or more pages of sideways RAM may be treated as a contiguous block of memory, referenced by means of the appropriate addresses.

The arrangement of the four pages is shown in the diagram below.

<rom id>s 4, 5, 6 and 7 relate specifically to the memory map arrangement in the Master Series computers; W, X, Y and Z are not machine-specific and should be used if software is also to be used on previous BBC microcomputers fitted with the sideways RAM upgrade.

Note that when using the pseudo-addressing system, 16 bytes at the start of each bank of sideways RAM are allocated for use with ROM header information – use of the complete block of four banks of sideways RAM for data will therefore provide 65472 (rather than 65536 (64K)) bytes.

A number of commands are provided for the purpose of accessing and using the available sideways RAM. Those described below may be used in the command mode of the current language or in programs written in languages with access to the OSCLI routine.

Command	**Function**
`*SRDATA`	Allocate a bank of sideways RAM for use with pseudo addressing
`*SRLOAD`	Load a file into an area of sideways RAM
`*SRREAD`	Copy an area of sideways RAM into the main memory map
`*SRROM`	Allocate a bank of sideways RAM for use with absolute addressing
`*SRSAVE`	Save a designated area of sideways RAM
`*SRWRITE`	Copy an area of the main memory map to a bank of sideways RAM.

These commands are described in the following sections and the syntax descriptions make use of the following definitions:

<object>	a file associated with the current Filing System (as defined previously)
<start address>	a location in the main memory map, (in hexadecimal)
<end address>	a location in the main memory map, (in hexadecimal)
<sideways start>	a location in the sideways RAM, (in hexadecimal)
<sideways end>	a location in the sideways RAM, (in hexadecimal)
<rom id>	sideways RAM bank reference (see diagram)

`Q` (Quickly) – an optional parameter allowing the speed of certain commands to be increased.

Sideways RAM occupies the same addresses in the main memory map as the current Filing System hence any transfers involving both the Filing System and sideways RAM will require each to be paged in and out of memory as required, resulting in a noticeably slower performance than, say with a transfer to or from normal RAM.

Inclusion of the `Q` parameter designates all normal RAM between OSHWM and the bottom of the screen memory as a buffer, thereby speeding up the transfer in question.

Note, however, that the content of normal RAM between OSHWM and the bottom of the screen memory will be overwritten.

Note

For economy of space, the code for the commands listed above is contained in the Disc Filing System ROM (ROM number 9 (&09) – itself a part of the 1Mbit ROM). The commands will therefore be unavailable if ROM 9 is `*UNPLUG`ged.

Sideways RAM command descriptions

*SRDATA

Syntax

`*SRDATA` <rom id>

Purpose

To allocate a bank of sideways RAM for use with data.

Description

`*SRDATA` indicates that the specified bank of sideways RAM may be accessed using the pseudo addressing mode.

Note that a `*SRDATA` command must be issued for each bank to be put to this use.

Execution of a `*ROMS` command after one or more `*SRDATA` commands will show the specified ROM numbers as:

`ROM` <rom number> `RAM FF`

*SRLOAD

Syntax

`*SRLOAD` <object> <sideways start> [<rom id>] [`Q`]

Purpose

To load a specified file into a designated area of sideways RAM.

Description

This command functions in the same way as the normal `*LOAD` command with the exception that the area of memory into which the file is read is resident in the sideways RAM.

<sideways start> is either:

– an absolute address

  (normally in the range &8000 – &BFFF), in which case the <rom id> parameter must be specified in order to identify the bank of sideways RAM.

– a pseudo address

  (in the range &0 – &FFBF) which specifies the bank(s) implicitly and which does not require a <rom id> parameter.

For loading a ROM image into sideways RAM, the syntax of the command becomes:

`*SRLOAD` <object> `8000` <rom id> [`Q`]

Note that ROM images so loaded will not be initialised until a hard break (CTRL+BREAK) is executed.

*SRREAD

Syntax

`*SRREAD` <start address> <end address> <sideways start> [<rom id>]
or
`*SRREAD` <start address>`+`<length> <sideways start> [<rom id>]

Purpose

To copy an area of sideways RAM into normal memory.

Description

<start address> and <end address> relate to normal memory and define the number of bytes to be transferred.

<sideways start> is either:

– an absolute address

  (normally in the range &8000 – &BFFF), in which case the <rom id> parameter must be specified in order to identify the bank of sideways RAM.

– a pseudo address

  (in the range &0 – &FFBF) which specifies the bank implicitly and which does not require a <rom id> parameter.

*SRROM

Syntax

`*SRROM` <rom id>

Purpose

To allocate a bank of sideways RAM for use with absolute addressing.

Description

All banks of sideways RAM are allocated as ROM on power-up and, therefore, `*SRROM` need only be used to indicate a change in use for a particular page.

Execution of a `*ROMS` command following one or more `*SRROM` commands will show the specified ROM numbers as:

`ROM` <rom id> `ROM FF`

*SRSAVE

Syntax

`*SRSAVE` <object> <sideways start> <sideways end> [<rom id>] [`Q`]
or
`*SRSAVE` <object> <sideways start>`+`<length> [<rom id>] [`Q`]

Purpose

To save a designated bank of sideways RAM.

Description

<sideways start> and <sideways end> may be either:

– absolute addresses

(normally in the range &8000 – &BFFF), in which case the <rom id> parameter must be specified in order to identify the bank of sideways RAM.

– pseudo addresses

(in the range &0 – &FFBF) which specify the bank implicitly and which do not require a <rom id> parameter.

The use of an appropriate <sideways end> specifies the number of bytes to be transferred implicitly – it may be specified explicitly using the alternative syntax.

*SRWRITE

Syntax

`*SRWRITE` <sideways address> <end address> <sideways start> [<rom id>]
or
`*SRWRITE` <sideways address>`+`<length> <sideways start> [<rom id>]

Purpose

To copy an area of main memory to sideways RAM.

Description

`*SRWRITE` copies the specified block of main memory to an area of sideways RAM. <sideways address> always specifies the address of the first byte to be transferred; the number of bytes may be specified implicitly (using an <end address> parameter), or explicitly using `+`<length>.

<sideways start> may be either:

– an absolute address

(normally in the range &8000 – &BFFF), in which case the <rom id> parameter must be specified in order to identify the bank of sideways RAM.

– a pseudo address

(in the range &0 – &FFBF) which specifies the bank implicitly and which does not require a <rom id> parameter.

Using sideways RAM from assembly language

OSBYTE calls 68 (Test RAM presence) and 69 (Test pseudo/absolute use of bank) are described in section D.

Two additional OSWORD calls are provided and are described below. On entry, A contains the call number and X and Y point to a control block in memory (details of which are provided in the descriptions below).

OSWORD 66 (&42) Block transfer to/from sideways RAM

OSWORD 66 carries out the functions of the `*SRREAD` and `*SRWRITE` commands.

Parameter block size : 10

Parameter block format : XY : bit 7 clear for reading from sideways RAM
bit 7 set for writing to sideways RAM
bit 6 clear for absolute addressing mode
bit 6 set for pseudo addressing mode
bits 0–5 reserved (set to zero)

XY+1 = <LSB of start address>
XY+2 .
XY+3 .
XY+4 = <MSB of start address>
XY+5 = <LSB of block length>
XY+6 = <MSB of block length>
XY+7 = <rom id> (for absolute addressing only)
XY+8 = <LSB of sideways address>
XY+9 = <MSB of sideways address>

On exit : The parameter block remains unchanged.

Note

<rom id>s W,X,Y and Z are denoted by &10 – &13 respectively.

OSWORD 67 (&43) Load or save to/from sideways RAM

OSWORD 67 carries out the functions of the `*SRLOAD` and `*SRSAVE` commands.

Parameter block size : 12

Parameter block format : XY : bit 7 clear for saving from sideways RAM
bit 7 set for loading to sideways RAM
bit 6 clear for absolute addressing mode
bit 6 set for pseudo addressing mode
bits 0–5 reserved (set to zero)

XY+1 = <LSB of file name address> (in main memory)
XY+2 = <MSB of file name address>
XY+3 = <rom id> (absolute addressing only)
XY+4 = <LSB of start address>
XY+5 = <MSB of start address>
XY+6 = <LSB of file length> (save operations only)
XY+7 = <MSB of file length>
XY+8 = <LSB of buffer start address> (see note)
XY+9 = <MSB of buffer start address>
XY+10 = <LSB of buffer length> (see note)
XY+11 = <MSB of buffer length>

On exit : The parameter block is unchanged unless the specification of the buffer causes it to be overwritten during the transfer.

Notes

The content of parameter block bytes XY+10 and XY+11 specifies the length of the buffer to be used for the transfer:

– If this value is zero, the transfer is carried out using the default buffer and any buffer start address specified in XY+8 and XY+9 is ignored. (This is equivalent to the execution of a `*SRLOAD` or a `*SRSAVE` command without a `Q` parameter).

– If the value specified in XY+10 and XY+11 is non-zero (but less than 32768), the specified number of bytes are used starting at the address specified in XY+8 and XY+9. (This option is not provided in either `*SRLOAD` or `*SRSAVE`).

– If the value specified in XY+10 and XY+11 exceeds 32768, the transfer is carried out using locations between OSHWM and the bottom of the screen memory for the current mode – any start address specified in XY+8 and XY+9 is ignored. (This option is equivalent to a `*SRLOAD` or a `*SRSAVE` command with a `Q` parameter).

<rom id>s W, X, Y and Z are denoted by &10 – &13 respectively.

Note that there is no direct assembly language equivalent of either `*SRROM` or `*SRDATA` – if required they must be executed by means of a call to the OSCLI routine (see section D.4).

H The Cassette Filing System (CFS)

H.1 Introduction

The Cassette Filing System (CFS) is the cheapest filing system to use with the computer – all that is needed to use it are an ordinary domestic cassette tape recorder and audio tapes.

Because it uses such simple hardware, the Cassette Filing System is comparatively primitive, not providing facilities such as directories or random access files. It is, however, very powerful compared to the tape filing facilities found in other low-cost microcomputer systems.

The CFS operates at two speeds:

– 1200 baud, selected by the command `*TAPE` (or `*TAPE12`);
– 300 baud, selected by the command `*TAPE3`.

The slower speed is provided for increased reliability with poor quality tape recorders and/or tapes. Most recorders will operate well at 1200 baud, especially if proper 'computing quality' tapes are used.

The CFS provides a degree of 'motor control'. That is, if the recorder has an input to turn the motor on and off (often found close to the microphone input), the computer can control this remotely. The correct cassette lead must be used for this facility to work; see the Welcome Guide supplied with the computer.

H.2 CFS command summary

A summary of the commands applicable to the Cassette Filing System is given below. Note that all the commands are 'common', (ie recognised by the MOS) – there are no 'intrinsic' commands. Full descriptions of each command may therefore be found in section G.5.

Command	Description
*BUILD	Send lines of keyboard input to a cassette file
*CAT	Display the names of files held on the current cassette
*CLOSE	Close a currently open (sequential) cassette file
*DUMP	Display a hex. and ASCII dump of a cassette file
*EX	Display file information for the files held on the current cassette
*EXEC	Take subsequent lines of input from a cassette file
*LIST	Display a cassette file (with line numbers)
*LOAD	Load a cassette file into memory
*OPT0	Restore default Filing System options
*OPT1	Specify reporting level during file operations
*OPT2	Specify action in the event of cassette loading errors
*OPT3	Specify interblock gap length for file output
*PRINT	Display a cassette file in pure ASCII
*RUN	Load and execute a machine code program from cassette
*SAVE	Save a block of memory to a cassette file
*SPOOL	Send all VDU output to a cassette file
*TYPE	Display a cassette file (without line numbers)

H.3 Using the CFS from assembly language

The routines described below are those which are either wholly (or partially) implemented for the Cassette Filing System. For those which are only partially implemented, only those facilities relevant to the Cassette Filing System are described – see sections I–J for descriptions applicable to other Filing Systems.

OSFIND — Open or close a file for byte access

Call address &FFCE
Indirected via &21C (PINDV)

On entry : A defines the action to be taken.

X and/or Y contain values depending upon the action specified.

Actions specified by A :

A = 0 (&00) indicates that a file is to be closed.

Under Filing Systems other than the Cassette Filing System, Y may contain either the <file handle> of the file to be closed or zero, which indicates that all currently open sequential files associated with the current Filing System are to be closed. These values also apply to the Cassette Filing System although only one file may be open at a time.

A <file handle> is allocated by the Filing System when a file is opened.

A = 64 (&40) indicates that a file is to be opened for input.

X and Y point to the location in memory (X=LSB, Y=MSB) containing the first character of the file name (which must be terminated by a carriage return character).

The file name may be null (ie merely a carriage return character), in which case the next file encountered will be opened.

A = 128 (&80) indicates that a file is to be opened for output.

X and Y point to the file name as described above.

On exit : X and Y are preserved.

A is preserved if zero on entry, otherwise A contains the file handle allocated to the file.

C, N, D and V are undefined.

The interrupt state is preserved but may be enabled during the operation.

OSGBPB — Read or write a group of bytes

Call address &FFD1
Indirected via &21A (GBPBV)

On entry : A defines the action to be taken.

X = <LSB of parameter block address>
Y = <MSB of parameter block address>

Parameter block size : 9

Parameter block format :

XY	=	<file handle>
XY+1	=	<LSB of pointer to data in memory>
XY+2		.
XY+3		.
XY+4	=	<MSB of pointer to data in memory>
XY+5	=	<LSB of number of bytes to transfer>
XY+6		.
XY+7		.
XY+8	=	<MSB of number of bytes to transfer>

Actions specified by A :

A = 2 (&02) Append bytes to the file.

The number of bytes specified in XY+5 to XY+8, starting at the address specified in XY+1 to XY+4 are appended to the specified file (ie the bytes are written to the file starting at the current value of the file pointer, which is incremented for each byte transferred).

A = 4 (&04) Read bytes from the current position in the file.

The number of bytes specified in XY+5 to XY+8 are transferred from the specified file and stored in memory locations starting at the address specified in XY+1 to XY+4. Reading commences from the current value of the file pointer, which is incremented for each byte transferred.

On exit : A, X and Y are preserved.

N, V and Z are undefined.

C = 0 indicates that the transfer was completed

C = 1 indicates that the transfer was incomplete for some reason.

The values in the parameter block are updated to reflect the position after the transfer (ie XY+1 to XY+4 contain the address of the byte after the last byte transferred to or from the file and XY+5 to XY+8 contain the number of bytes remaining to be transferred – zero if C = 0.

The interrupt state is preserved, but may be enabled during the call.

OSBPUT — Write a byte to an open file

Call address &FFD4
Indirected via &218 (BPUTV)

On entry : Y = <file handle>

A = <byte to be written>

The byte is written to the specified file using the current value of the file pointer, which is incremented for each byte written to the file.

On exit : A, X and Y are preserved.

C, N, V and Z are undefined.

The interrupt state is preserved but may be enabled during the call.

OSBGET — Read a single byte from an open file

Call address &FFD7
Indirected via &216 (BGETV)

On entry : Y = <file handle>

The byte at the current value of the file pointer is read from the file. The file pointer is incremented for each byte read.

On exit : A = <byte read>

X and Y are preserved.

N, V and Z are undefined.

C = 0 indicates that the transfer was completed
C = 1 indicates that end of file was encountered and that the value in A should be discarded.

The interrupt status is preserved but may be enabled during the call.

OSARGS — Read filing system information

Call address &FFDA
Indirected via &214 (ARGSV)

On entry : A specifies the action to be taken.

X and/or Y contain values depending on A.

Actions specified by A :

A = 0 (&00)	Y = 0	Return Filing System number
	Y = <file handle>	Return sequential pointer
A = 1 (&01)	Y = 0	Return address of remainder of the last command line

(other combinations of values for A and Y have no effect under the Cassette Filing System).

On exit : X and Y are preserved.

A is preserved except when A=0 and Y=0 on entry, in which case it contains the Filing System number (see section G.1).

If A=1 and Y=0 on entry, the address of the remainder of the last command line is returned in a four-byte zero page block pointed to by X. This address always points to the IO processor and should therefore be read using OSWORD 5. The text making up the remainder of the last command line always terminates with a carriage return character.

The interrupt status is preserved although interrupts may be enabled during the call.

OSFILE — Load or save a complete file

Call address &FFDD
Indirected via &212 (FILEV)

On entry : A specifies the action to be performed

X = <LSB of parameter block address>
Y = <MSB of parameter block address>

Parameter block size : 18

Parameter block format :

XY = <LSB of address of file name>
XY+1 = <MSB of address of file name>
XY+2 = <LSB of load address for file>
XY+3 .
XY+4 .
XY+5 = <MSB of load address for file>
XY+6 = <LSB of execution address for file>
XY+7 = .
XY+8 = .
XY+9 = <MSB of execution address for file>
XY+10 = LSB of either <start address> or <length>
XY+11 = .
XY+12 = .
XY+13 = MSB of either <start address> or <length>
XY+14 = <LSB of end address>
XY+15 = .
XY+16 = .
XY+17 = <MSB of end address>

The file name pointed to by XY and XY+1 must terminate with a carriage return.

Actions specified by A :

A = 0 (&00) Save a block of memory using the information provided in the control block (XY+10 to XY+13 contain the start address of the data in memory, XY+14 to XY+17 contain the address of the byte after the last to be saved).

A = 255 (&FF) Load the named file into memory at a location determined by the content of parameter block byte XY+6:

If XY+6 is zero, the file is loaded into memory at the address specified in XY+2 to XY+5.

IF XY+6 is non-zero, the file is loaded into memory using the files own load address (see section H.5).

The file name may be null (ie merely a carriage return).

On exit : A is undefined.

X and Y are preserved.

C, N, V and Z are undefined.

The interrupt status is preserved but may be enabled during the call.

H.4 Error messages

The following error messages may be produced by the Cassette Filing System or by other components of the system during cassette operations:

`Bad address` (Error 252)

Wherever load and execution addresses of files are specified, they should lie in the range 0 to FFFFFFFF. No ampersand is required or allowed before the number.

An address which is not a hexadecimal number in the range given above will yield a `Bad Address` message.

`Bad command` (Error 254)

This error is given when a command is not recognised by any of the MOS, the Cassette Filing System and the library filing system (if selected).

`Block?` (Error 218)

Files stored by the Cassette Filing System are divided into blocks which are numbered sequentially from zero, and usually contain between zero and 256 data bytes.

This error is generated if a block is encountered whose number is not one greater than the previous block – the user is given the chance to try again by rewinding the tape (unless the `*OPT2` setting causes the load to abort on error).

`Channel` (Error 222)

This error can be produced when accessing sequential files, either because the <file handle> parameter specified does not belong to a currently open file or because an attempt has been made to read from the cassette file opened for output.

`Data?` (Error 216)

Each block of data in files stored by the Cassette Filing System contains a 'cyclic redundancy check' (CRC) value (stored as two bytes) which enables almost all transmission errors to be detected.

This error occurs when the calculated CRC for a block of data is not the same as the one stored at the end of the data block.

`EOF` (Error 223)

This error is generated if an attempt is made to read past the last byte of a file.

It is possible to check for the end of file condition before an OSBGET by using OSBYTE &7F with the <file handle> in X; if X is non-zero on exit, the end of the file has been reached.

`File?` (Error 219)

Block headers in files stored by the Cassette Filing System contain, among other things, the name of the file. This error is produced if, when accessing a file, the Filing System encounters a file name which is not identical to that contained in the previous block.

`Locked` (Error 213)

There is provision for CFS files to be made 'execute only', so that they may be `*RUN` but not `*LOAD`ed. An attempt to `*LOAD` or `LOAD` a locked CFS file will give this error message.

Error messages in numerical order

Error number	Message
213 (&D5)	`Locked`
216 (&D8)	`Data?`
218 (&DA)	`Block?`
219 (&DB)	`File?`
222 (&DE)	`Channel`
223 (&DF)	`EOF`
252 (&FC)	`Bad address`
254 (&FE)	`Bad command`

H.5 Technical information

The Cassette Filing System records data in standard CUTS (Kansas City) format at 300 baud and in modified CUTS at 1200 baud.

Files are stored as one or more blocks; the file is preceded by 5 seconds of 2400Hz 'leader' tone, and blocks are separated by an inter-block gap of the same frequency. The duration of the interblock gap is fixed for 'whole file' operations but may be specified by the user for output using byte access (see `*OPT`, section G.5).

The format of a block is:

Sync byte	(ASCII &2A or "*")
Filename	(1–10 characters, terminated by a 0 byte)
Load address	(four bytes, LSB first)
Execution address	(four bytes, LSB first)
Block number	(two bytes, LSB first)
Number of data bytes	(two bytes, LSB first)
Block status flag	(one byte (see below))
Spare	(four bytes, all zero)
CRC on header	(two bytes, MSB first)
Data	(0 to 256 bytes)
CRC on data	(two bytes, MSB first)

Three bits of the block status byte are used by the Cassette Filing System:

- bit 0 set if the file may only be `*RUN`;
- bit 6 set if the data length of the block is zero;
- bit 7 set if the block is the last in the file.

I The ROM Filing System (RFS)

I.1 Introduction

The ROM Filing System (RFS) is a very cheap filing system to use for loading small to medium-sized utilities and games, from cartridges or paged ROMs. It is much quicker in operation than the Cassette Filing System, but cannot be used to save information.

Files used by the ROM Filing System are stored in ROM (or EPROM) chips, either inserted in the vacant sockets on the main printed circuit board or, more commonly, mounted in ROM cartridges for insertion into either of the two cartridge ROM sockets positioned above the numeric keypad.

See section F for information concerning the ROM numbers allocated to the various sockets.

Such ROMs contain service routines that enable the filing system to read bytes from files inside the ROM. ROM Filing System files are segmented into blocks in a similar way to CFS files but, in order to save valuable space, only the first and last blocks need header information.

You will most probably use the ROM Filing System with commercially-produced ROM cartridges but, if you have access to the necessary tools to program your own EPROMs, you can use it more flexibly. For example:

- a BASIC programmer could keep a library of procedures on ROM, to merge with programs he is developing;
- a wordprocessor operator could keep standard letters or paragraphs on ROM;
- machine code programmers could create ROM files containing standard utilities such as disassemblers and memory dump routines.

The ROM Filing System is particularly useful if you use cassette as your main storage medium – the Disc and Advanced Disc Filing Systems are much faster and more versatile and the ROM Filing System will therefore be less attractive. However, the ROM Filing System may still be worth using as a library to hold commonly-used routines, since these will then be available regardless of what discs you are using on the drive. In this case, you should select one or other of the Disc Filing Systems as your current filing system, and the ROM Filing System as the library filing system, using the command:

```
*-ROM-LIBFS
```

Notes

1. The ROM Filing System is used if you try to auto-boot the Cassette Filing System using SHIFT+BREAK; this causes the MOS to try to `*RUN` an RFS file called `!BOOT` and, if no such file exists on the ROM Filing System, the CFS will be entered.
2. If you press BREAK with the ROM Filing System selected as current filing system, the CFS is selected. This differs from the usual action, where the previous filing system is reselected after BREAK.

I.2 RFS command summary

A summary of the commands applicable to the ROM Filing System is given below. Note that as with the Cassette Filing System, all the commands are 'common', (ie recognised by the MOS) – there are no 'intrinsic' commands. Full descriptions of each command may therefore be found in section G.5.

Command	Description
*CAT	Display the names of files held in ROM
*CLOSE	Close a currently open (sequential) ROM input file
*DUMP	Display a hex. and ASCII dump of a ROM file
*EX	Display file information for the files held in ROM
*EXEC	Take subsequent lines of input from file in ROM
*LIST	Display a ROM file (with line numbers)
*LOAD	Load a ROM file into memory
*OPT0	Restore default Filing System options
*OPT1	Specify reporting level during file operations
*PRINT	Display a ROM file in pure ASCII
*RUN	Load and execute a machine code program from ROM
*TYPE	Display a ROM file (without line numbers)

I.3 Using the RFS from assembly language

The routines described below are those which are either wholly (or partially) implemented for the ROM Filing System. For those which are only partially implemented, only those facilities relevant to the ROM Filing System are described – see sections H.3 and J.11 for descriptions applicable to other Filing Systems.

OSFIND — Open or close a file for byte access

Call address &FFCE
Indirected via &21C (FINDV)

On entry : A defines the action to be taken

X and/or Y contain values depending upon the action specified.

Actions specified by A :

A = 0 (&00) indicates that a file is to be closed

Under Filing Systems other than the ROM Filing System, Y may contain either the <file handle> of the file to be closed or zero, which indicates that all currently open sequential files associated with the current Filing System are to be closed. These values also apply to the ROM Filing System although only one file may be open at a time (and that open for input only).

A <file handle> is allocated by the Filing System when a file is opened.

A = 64 (&40) indicates that a file is to be opened for input

X and Y point to the location in memory (X=LSB, Y=MSB) containing the first character of the file name (which must be terminated by a carriage return character).

The file name may be null (ie merely a carriage return character), in which case the next file encountered will be opened.

On exit : X and Y are preserved

A is preserved if zero on entry, otherwise A contains the file handle allocated to the file

C, N, D and V are undefined

The interrupt state is preserved but may be enabled during the operation.

OSGBPB — Read a group of bytes

Call address &FFD1
Indirected via &21A (GBPBV)

On entry : A defines the action to be taken (although only one value is appropriate)

X = <LSB of parameter block address>
Y = <MSB of parameter block address>

Parameter block size : 9

Parameter block format :

XY	= <file handle>
XY+1	= <LSB of pointer to data in memory>
XY+2	.
XY+3	.
XY+4	= <MSB of pointer to data in memory>
XY+5	= <LSB of number of bytes to transfer>
XY+6	.
XY+7	.
XY+8	= <MSB of number of bytes to transfer>

Actions specified by A :

A = 4 (&04) Read bytes from the current position in the file

The number of bytes specified in XY+5 to XY+8 are transferred from the specified file and stored in memory locations starting at the address specified in XY+1 to XY+4. Reading commences from the current value of the file pointer, which is incremented for each byte transferred.

On exit : A, X and Y are preserved

N, V and Z are undefined

C = 0 indicates that the transfer was completed

C = 1 indicates that the transfer was incomplete for some reason.

The values in the parameter block are updated to reflect the position after the transfer (ie XY+1 to XY+4 contain the address of the byte after the last byte transferred to or from the file and XY+5 to XY+8 contain the number of bytes remaining to be transferred – zero if C = 0).

The interrupt state is preserved, but may be enabled during the call.

OSBGET Read a single byte from an open file

Call address &FFD7
Indirected via &216 (BGETV)

On entry : Y = <file handle>

The byte at the current value of the file pointer is read; the file pointer is incremented for each byte read.

On exit : A = <byte read>

X and Y are preserved

N, V and Z are undefined

C = 0 indicates that the transfer was completed

C =1 indicates that end of file was encountered and that the value in A should be discarded.

The interrupt status is preserved but may be enabled during the call.

OSARGS Read filing system information

Call address &FFDA
Indirected via &214 (ARGSV)

On entry : A specifies the action to be taken

X and/or Y contain values depending on A

Actions specified by A :

A = 0 (&00)	Y = 0	Return Filing System number
	Y = <file handle>	Return sequential pointer
A = 1 (&01)	Y = 0	Return address of remainder of the last command line.

(other combinations of values for A and Y have no effect under the ROM Filing System).

On exit : X and Y are preserved

A is preserved except when A=0 and Y=0 on entry, in which case it contains the Filing System number (see section G.1).

If A=1 and Y=0 on entry, the address of the remainder of the last command line is returned in a four-byte zero page block pointed to by X. This address always points to the IO processor and should therefore be read using OSWORD 5. The text making up the remainder of the last command line always terminates with a carriage return character.

The interrupt status is preserved although interrupts may be enabled during the call.

OSFILE — Load or save a complete file

Call address &FFDD
Indirected via &212 (FILEV)

On entry : A specifies the action to be performed (although only one value is appropriate)

X = <LSB of parameter block address>
Y = <MSB of parameter block address>

Parameter block size : 18

Parameter block format :

XY = <LSB of address of file name>
XY+1 = <MSB of address of file name>
XY+2 = <LSB of load address for file>
XY+3 .
XY+4 .
XY+5 = <MSB of load address for file>
XY+6 = <LSB of execution address for file>
XY+7 .
XY+8 .
XY+9 = <MSB of execution address for file>

XY+10 = LSB of either <start address> or <length>
XY+11 .
XY+12 .
XY+13 = MSB of either <start address> or <length>
XY+14 = <LSB of end address>
XY+15 .
XY+16 .
XY+17 = <MSB of end address>

The file name pointed to by XY and XY+1 must terminate with a carriage return.

Action specified by A :

A = 255 (&FF) Load the named file into memory at a location determined by the content of parameter block byte XY+6:

If XY+6 is zero, the file is loaded into memory at the address specified in XY+2 to XY+5.

If XY+6 is non-zero, the file is loaded into memory using the files own load address (see section I.5).

The file name may be null (ie merely a carriage return)

On exit : A is undefined

X and Y are preserved

C, N, V and Z are undefined

The interrupt status is preserved but may be enabled during the call.

I.4 Error messages

The following errors may be generated by the ROM Filing System or by other components of the system during ROM operations.

`Bad Address` (Error 252)

Wherever load and execution addresses of files are specified, they should lie in the range 0 to FFFFFFFF. No ampersand is required or allowed before the number.

An address which is not a hexadecimal number in the range given above will yield a `Bad address` error.

`Bad command` (Error 254)

This error is given when a command is not recognised by any of the MOS, the ROM Filing System and the library filing system (if selected).

`Bad ROM` (Error 215)

When a ROM Filing System service ROM is used, the files it contains must conform to a given format (see section I.5).

This error is generated if an attempt is made to use an file which does not conform to the standard.

`Channel` (Error 222)

Under the ROM Filing System, this error is produced if an attempt is made to read bytes from a file which has not been opened.

`EOF` (Error 223)

This error is generated if an attempt is made to read past the last byte of a file.

It is possible to check for the end of file condition before an OSBGET by using OSBYTE &7F with the <file handle> in X; if X is non-zero on exit, the end of the file has been reached.

`Not found` (Error 214)

This occurs in the ROM Filing System when an attempt is made to `OPENIN` or `LOAD` a non-existent file.

`Locked` (Error 213)

There is provision for RFS files to be made 'execute only', so that they may be `*RUN` but not `*LOAD`ed. An attempt to `*LOAD` or `LOAD` locked RFS file will give this error message.

Errors in numerical order

Error number	Message
213 (&D5)	Locked
214 (&D6)	File not found
215 (&D7)	Bad ROM
222 (&DE)	Channel
223 (&DF)	EOF
252 (&FC)	Bad address
254 (&FE)	Bad command

I.5 Technical information

Files are stored under the ROM Filing System in essentially the same format as for cassette files. For the first and last blocks of a file, the format is:

Sync byte	(ASCII &2A or "*")
Filename	(1–10 characters, terminated by a 0 byte)
Load address	(four bytes, LSB first)
Execution address	(four bytes, LSB first)
Block number	(two bytes, LSB first)
Number of data bytes	(two bytes, LSB first)
Block status flag	(one byte (see below))
End of file address	(four bytes (see below))
CRC on header	(two bytes, MSB first)
Data	(0 to 256 bytes)
CRC on data	(two bytes, MSB first)

Three bits of the block status byte are used by the ROM Filing System:

– bit 0 set if the file may only be `*RUN`
– bit 6 set if the data length of the block is zero
– bit 7 set if the block is the last in the file

The end of file address is the address of the byte after the end of the file in ROM – it is used for 'skipping' unwanted files during searches.

Blocks other than the first and the last may have their header information abbreviated to a single byte containing a "#" character (&23). If the header is abbreviated in this way, it is assumed to be the same as for the previous block.

The character "+" (&2B) is used as an 'end of ROM' marker. Note that it is possible for a file to span over two ROMs although care must be taken to ensure that the block number sequence is maintained in the next ROM to be read.

J The Disc Filing Systems (DFS and ADFS)

J.1 Introduction

The two standard Disc Filing Systems provide users with fast, high-capacity, convenient access to files. The choice between the two is largely one of personal preference, although the Advanced Disc Filing System is more sophisticated (and must be used if a Winchester disc drive is connected to the computer).

Users of earlier disc-based BBC Microcomputers may choose to continue using the Disc Filing System although it is a relatively simple matter to transfer files to discs formatted for use with the Advanced Disc Filing System.

Descriptions of the commands available under the Disc and Advanced Disc Filing Systems are given in the following sections; the information immediately below is of a general nature and relates to both Filing Systems.

Floppy (flexible) discs

Conventional 5.25″ floppy discs consist of a thin, circular disc of flexible plastic coated with a magnetic material. The disc itself is enclosed inside a rectangular casing with apertures to provide access for the drive hub and the read/write heads. A notch is normally cut into one side of the casing; if the notch is uncovered, data may be written to the disc; if the notch is covered (using a small, self-adhesive tab), the disc is said to be 'write protected' and data may only be read from the disc.

Most discs are provided with paper envelopes which are designed to protect the exposed surfaces of the disc and it is strongly recommended that these are used whenever a disc is not in use. Discs should also be kept away from magnetic fields (such as those produced by televisions, monitors (and computer power supplies!)) and should not be subjected to extremes of temperature.

New discs contain no data and may not be used with the computer until they have been formatted for use with the type of disc drive and the appropriate Filing System:

– The Disc Filing System provides facilities for using either 40- or 80-track discs (ie single or double track density corresponding to either 48 or 96 tracks per inch), in either single- or double-sided configurations. Formatting (and format verification) are carried out using the `*FORM` and `*VERIFY` commands (see section J.3).

The Disc Filing System uses a single data density recording format which provides either 100K or 200K bytes per disc surface.

See `*CONFIGURE FDRIVE` (in section C) for instructions relating to the means of specifying the default floppy disc controller parameters.

– The Advanced Disc Filing System provides facilities for using either single-sided 40-track discs or single- or double-sided 80 track discs. Formatting (and format verification) are carried out using the AFORM and VERIFY utilities provided on the Welcome disc supplied with the computer.

  IT IS ESSENTIAL THAT YOU DO NOT LOSE OR DAMAGE THE WELCOME DISC AS YOU WILL BE UNABLE TO FORMAT DISCS FOR USE WITH THE ADVANCED DISC FILING SYSTEM.

  The Advanced Disc Filing System uses a double data density recording format which gives a single-sided 40-track disc a capacity of approximately 160K bytes and a single-sided 80-track disc approximately 320K. Both surfaces of a double-sided 80-track disc are treated as a single entity under the Advanced Disc Filing System, in which case the capacity is approximately 640K bytes.

Winchester discs

Winchester discs are rigid metal discs fully and permanently encased in a drive housing – user access to the disc itself is normally prevented and not recommended.

Winchester disc units manufactured by Acorn Computers Limited are formatted for use with the Advanced Disc Filing System prior to purchase – they are not suitable for use with the conventional Disc Filing System. Units produced by other manufacturers are unlikely to be compatible with the Advanced Disc Filing System.

Other types of disc

The standard interfaces are able to control a variety of different types of disc drive (typically 3.5″ mini-disc drives) provided that suitable connecting cables are available. You will need to consult the manufacturer's documentation to determine the track and data density possibilities.

J.2 DFS command summary

A summary of the commands available under the Disc Filing System is given below. Commands marked 'common' under the heading 'type' are described in section G.5; those marked 'intrinsic' are specific to the Disc Filing System and are described in the next section.

Command	Function	Type
*ACCESS	Lock (or unlock) a file or group of files	Intrinsic
*APPEND	Append subsequent keyboard input to a file	Common
*BACKUP	Copy a complete disc surface	Intrinsic
*BUILD	Send subsequent lines of keyboard input to a file	Common
*CAT	Display a file catalogue	Common
*CLOSE	Close all currently open sequential disc files	Common
*COMPACT	Reorganise free space on a disc surface	Intrinsic
*COPY	Copy a file from one disc surface to another	Intrinsic
*CREATE	Reserve space for a file	Common
*DELETE	Delete the name of a file from the current catalogue	Common
*DESTROY	Delete the name(s) of a file or group of files	Intrinsic
*DIR	Set current directory	Intrinsic
*DISC	Select the Disc Filing System	Intrinsic
*DRIVE	Set current drive number	Intrinsic
*DUMP	Display a hex. and ASCII dump of a file	Common
*ENABLE	Enable use of certain commands	Intrinsic
*EX	Display Filing System information for files	Common
*EXEC	Take subsequent lines of input from a file	Common
*FORM	Format disc(s) for use with DFS	Intrinsic
*FREE	Display free space information	Intrinsic
*INFO	Display information about a file or group of files	Common
*LIST	Display the content of a file (with line numbers)	Common
*LIB	Set current library directory	Intrinsic
*LOAD	Load file into memory	Common
*MAP	Display free space map	Intrinsic
*OPT0	Restore default Filing System options	Common
*OPT1	Set reporting level during file operations	Common
*OPT4	Set auto-start option	Common
*PRINT	Display a file in pure ASCII	Common
*REMOVE	Delete a file from the current catalogue	Common
*RENAME	Change the name of a file	Intrinsic
*RUN	Load and execute a machine code program	Common
*SAVE	Save a block of memory as a file	Common

*SPOOL	Send all subsequent VDU output to a file	Common
*SPOOLON	Append all subsequent VDU output to a file	Common
*TITLE	Define a disc title	Intrinsic
*TYPE	Display the content of a file (without line numbers)	Common
*VERIFY	Verify the formatting on a disc	Intrinsic
*WIPE	Delete a file or group of files	Intrinsic

J.3 DFS command descriptions

This section contains descriptions of all intrinsic Disc Filing System commands – where appropriate, the minimum abbreviation is given with the command name. The syntax descriptions use the definitions given in section G.2.

*ACCESS *AC.

Syntax

`*ACCESS` <object> [`L`]

Purpose

To prevent a file (or group of files) from being deleted or overwritten.

Description

`*ACCESS` 'locks' or 'unlocks' the specified file depending upon the presence or absence of the optional `L` parameter.

Locked files may not be deleted, written to or overwritten (for example, if a locked file is loaded, it may not be saved again with the same name).

Note however, that locking does not prevent a file from being destroyed by means of `*FORM` or `*BACKUP` (see below).

The wildcard characters # and * may be used with `*ACCESS` in order to lock or unlock a group of files.

*BACKUP *BAC.

Syntax

`*BACKUP` <drive number> <drive number>

Purpose

To read all the information on one disc surface and write it to another, thereby producing two identical discs.

Description

`*BACKUP` copies the entire content of the first <drive number> to the second <drive number>; if the two parameters are the same it is assumed to mean that the copy is to be made in a single drive and the user is prompted to load the source and destination discs alternately.

All information previously on the destination disc is overwritten – if the source disc is blank, the destination disc will end up blank as well!

Unless it is immediately preceded by a *ENABLE command (see below), *BACKUP produces the prompt:

```
Go (Y/N) ?
```

This is provided merely as a safety measure.

Notes

1. The contents of memory may be overwritten by this command. A program or data currently in memory should therefore be saved before use of *BACKUP.
2. The speed at which *BACKUP operates is dependent upon the amount of memory which may be used as a buffer – optimal performance is achieved in either Mode 7 or any of the Shadow modes.

*COMPACT *COM.

Syntax

*COMPACT [<drive number>]

Purpose

To reorganise the free space on a disc so that it occupies a single, contiguous block after the content of any files.

Description

Attempting to save a program or file on to a disc may produce the message Disc Full if there is no single space available on the disc large enough for the content of the file. It may be that there is enough space in total, but that it is split into several small sections created by deleting files.

*COMPACT moves all current files on the current or specified drive so that they occupy a contiguous block occupying the lowest numbered tracks – as a consequence the free space is 'compacted' into a single block starting at the sector immediately after the last block of the last file and extending to the last sector on the disc.

Clearly, *COMPACT will have an effect only if there is space between the files.

Notes

1. The contents of memory may be overwritten by this command. A program or data currently in memory should therefore be saved before use of *COMPACT.
2. The speed at which *COMPACT operates is dependent upon the amount of memory which may be used as a buffer – optimal performance is achieved in either Mode 7 or any of the Shadow modes.

*COPY

Syntax

*COPY <drive number> <drive number> <object>

Purpose

To copy a named file or files from one disc surface to another.

Description

Whereas *BACKUP operates upon all the files on a disc surface, *COPY operates upon a single file (or a group of files specified using the wildcard characters # and *).

The files are copied from the first <drive number> to the second; with prompts as for *BACKUP if both drive numbers are the same. Note that in the case of *COPY, files are added to the catalogue of the destination drive and it is therefore possible to encounter Disc full or Catalogue full errors.

Notes

1. The contents of memory may be overwritten by this command. A program or data currently in memory should therefore be saved before use of *COPY.
2. The speed at which *COPY operates is dependent upon the amount of memory which may be used as a buffer – optimal performance is achieved in either Mode 7 or any of the Shadow modes.

*DESTROY *DES.

Syntax

*DESTROY <object>

Purpose

To remove specified files from a disc in a single action.

Description

The common command *DELETE is used to delete a single file; *DESTROY performs an identical function if a unique <object> is specified but the use of the wildcard characters (# and *) with this command allows a group of files to be deleted in one operation.

Unless it is immediately preceded by a *ENABLE command, *DESTROY displays the prompt:

```
Go (Y/N) ?
```

which gives the user an opportunity to abort the command. If the reply is Y (or y), a list of the currently unlocked files matching the specification of <object> is displayed and the prompt is then repeated. The files are deleted only if the reply to the second prompt is Y (or y).

*DIR

Syntax

*DIR [<drive specification>][<directory name>]

Purpose

To change the current directory to the drive/directory specified.

Description

By default, the current directory is set to drive 0, directory $ (ie :0.$).

*DIR allows either or both of the drive number and directory letter for the current directory to be altered. Note that a command such as:

```
*DIR :1.W
```

is equivalent in effect to the commands:

```
*DRIVE 1  (see below)
*DIR W
```

*DISC *DI.

Syntax

*DISC *or*
*DISK

Purpose

To select the Disc Filing System.

Description

*DISC selects the Disc Filing System and makes it the current Filing System.

If the command is the first use of the Filing System since a power-on (or if the Disc Filing System is set as the default Filing System), drive 0 and directory $ are selected as both the current and library directories by default. If the Disc Filing System has been used previously, the current drive/directory and library directory are set to those last in use.

*DRIVE *DR.

Syntax

`*DRIVE` <drive number> [<`40`/`80`>]

Purpose

To change the current drive.

Description

`*DRIVE` changes the current drive to that specified although the directory letter associated with the previous drive remains unchanged, as does the specification of the library directory.

The second parameter is applicable to users of 80-track disc drives ONLY:

– `*DRIVE` <drive number> `40`

  sets the current drive to <drive number> and instructs the disc controller to double-step the read-write heads so that the drive can read 40-track discs. Note that the action applies only to read operations – write operations will produce a `Disc read only` message.

– `*DRIVE` <drive number> `80`

  resets the current drive to <drive number> and restores normal 80-track operation.

*ENABLE *ENA.

Syntax

`*ENABLE`

Purpose

To enable the use of certain commands.

Description

As a security measure, the commands `*BACKUP`, `*DESTROY` and `*FORM` (see below) display the prompt:

```
Go (Y/N) ?
```

before execution commences. Immediate, prior use of `*ENABLE` suppresses this prompt – any intervening command nullifies its effect.

*FORM *FO.

Syntax

`*FORM <40/80>` [<drive number>]…

Purpose

To format a disc for use with the Disc Filing System.

Description

The first parameter specifies which of the 40- or 80-track formats is required – the choice is dictated by the type of drive (ie it is not possible to format a 40-track disc in an 80-track drive or vice versa).

If no <drive number> parameter is specified, the user is prompted for the drive; if more than one <drive number> is specified, the specified format is applied to each drive in turn.

`*FORM` issues a security prompt:

```
Go (Y/N) ?
```

which may be suppressed by the use of `*ENABLE`.

If formatting of a track fails at the first attempt, a `?` is displayed after the track number concerned. If the track fails to format after six attempts, formatting is abandoned and a disc fault reported.

Note that the formatting process uses (approximately) 3.5K bytes above OSHWM (usually `PAGE` in BASIC) so any current program or data may be corrupted and should be saved prior to the execution of a `*FORM` command.

*FREE *FR.

Syntax

`*FREE` [<drive number>]

Purpose

To display free space available on a disc.

Description

If no <drive number> is specified, the current drive is assumed; otherwise the free space information for the specified drive is displayed. The format of the display is:

```
<ff> Files <sss> Sectors <bbbbbbb> Bytes Free
<ff> Files <sss> Sectors <bbbbbbb> Bytes Used
```

where:

<ff>	is the number of files free/used in the catalogue in question (in decimal)
<sss>	is the number of sectors free/used for the content of files (in hexadecimal)
<bbbbbbb>	is the number of bytes free/used for the content of files (in decimal)

Note that the <sss> and <bbbbbbb> values are absolute (ie they do not take any fragmentation of the available free space into account).

*LIB

Syntax

`*LIB` [<drive specification>]<directory name>

Purpose

To set the library directory.

Description

By default, the library directory is drive 0, directory $ (ie **`:0.$`**, as for the current directory). `*LIB` is used to change this definition by reference to either a new drive and directory letter or merely a new directory letter (in which case the drive remains unchanged.)

Note that unlike `*DIR`, `*LIB` does not affect either the current drive or the current directory, it specifies the directory to be examined if the object of a `*`<file name> command cannot be located in the current directory. The search may be extended to include the current and library directories of the Library Filing System if required (see section G.5).

*MAP

Syntax

*MAP [<drive number>]

Purpose

To display a map of the free space available on a disc.

Description

Whereas *FREE gives an indication of the total amount of free space, *MAP displays a map of the distribution of the free space, in terms of start sector numbers and lengths, as shown below. If the <drive number> is omitted, the current drive is assumed.

```
Address  :  Length
<aaa>    :  <lll>
.           .
.           .
<aaa>    :  <lll>
```

where:

<aaa>	is a disc address (in hexadecimal)
<lll>	is the length of the free space entry (hexadecimal number of sectors)

The number of entries in the free space map displayed by *MAP is a good guide to the likely need for compaction.

*RENAME *REN.

Syntax

*RENAME [<directory specification>]<file name> [<directory specification>]<file name>

Purpose

To change the name (and/or directory) of a file.

Description

*RENAME changes the catalogue information relating to a specified file from that specified by the first parameter to that specified by the second. If either or both <directory specification>s are omitted, the current directory is assumed.

Note that it is not possible to rename a locked file and that the object defined by the second parameter must not already exist.

The syntax of *RENAME does not allow the drive number to be changed because this information is not held in the catalogue.

*TITLE *TIT.

Syntax

*TITLE <title string>

Purpose

To specify, or change a disc title.

Description

<title string> is a sequence of characters which will be written to the catalogue for the current drive and which will subsequently be displayed for every *CAT command.

The string is terminated by either the first space character (unless the whole string is enclosed in quotation marks) or the carriage return used to terminate the command. The disc title is either truncated or space filled (at the right) to twelve characters.

*VERIFY *V.

Syntax

*VERIFY [<drive number>]...

*WIPE *W.

Syntax

*WIPE <object>

Purpose

To delete specified files from the current catalogue.

Description

*WIPE operates in a similar manner to *DESTROY in that the specification of <object> may include the wildcard characters # and * to identify a group of files.

In the case of *WIPE, however, each entry matching the specification of <object> is displayed in turn and the user is prompted for the action to be taken – pressing Y (or y) causes the file to be deleted; depression of any other key causes the file to be retained.

J.4 DFS error messages

The following error messages may be produced by the Disc Filing System.

`Bad attribute` (Error 207)

This message signifies the use of any attribute other than `L` (or nothing) in a `*ACCESS` command.

`Bad dir` (Error 206)

This message signifies the use of an invalid directory name, for example one consisting of more than one character.

`Bad drive` (Error 205)

This message signifies that a <drive specification> is invalid, either because the `:` is missing or because the drive number is none of the characters 0, 1, 2 or 3.

`Bad name` (Error 204)

This message signifies the use of a file name which is invalid, for example too long, missing, or containing invalid characters.

`Bad option` (Error 203)

Only `*OPT0`, `*OPT1` and `*OPT4` commands are recognised by the Disc Filing System. This message indicates the use of any other option number.

`Can't Extend` (Error 191)

The Filing System maintains a record of the start sector and length (in bytes) of all files held in the current catalogue. These values indicate the 'extent' of the file on the disc – the data contained within the file may occupy only a small proportion of the available space. New data may always be written to the file (using commands such as `*APPEND` and `*SPOOLON` or by writing data to a file from a program) provided that the length of the data in the file does not exceed the extent of the file.

Once the limit is reached, the Filing system attempts to extend the file by allocating further sectors, which is possible only if the file is adjacent to an area of free space.

This message indicates that no adjacent free space is available to extend a file.

`Cat full` (Error 190)

Disc Filing System catalogues may contain a maximum of 31 file names. This message indicates that there is no more room for file names in the catalogue for the current (or specified drive), regardless of the free space available on the remainder of the disc.

`Channel` (Error 222)

This message indicates an attempt to access a file which has not been opened for byte access and for which, therefore, no file handle has been allocated by the Filing System.

`Disc changed` (Error 200)

This message is displayed if the Filing System detects that a disc has been changed while files on it are still open.

`Disc fault` <nn> `at :` <drive number> <tt>`/`<ss> (Error 199)

This message indicates that the Filing System was unable to read the disc surface associated with the current drive.

<nn> is the fault number (see section J.11)
<tt> is the track number
<ss> is the sector number on the specified track

One of the most common causes of this error is an attempt to access a 40-track disc in an 80-track drive or vice versa.

`Disc full` (Error 198)

In contrast to `Cat full`, which indicates that there is insufficient room to store the name of the file in the current catalogue, this message indicates that there is insufficient space to either open or save a file of the specified size.

Note that, by default, files opened using OSFIND &80 (BASICs OPENOUT) are allocated &4000 (16K) bytes.

`Disc read only` (Error 201)

This message indicates that an attempt has been made to write to a disc protected by means of a write protect tab or to write to an 80-track disc in 40-track mode.

`Drive fault` <nn> `at` <tt>`/`<ss> (Error 197)

This message indicates that the disc drive is not responding correctly to requests from the disc controller. The fault should be referred to your supplier.

`EOF` (Error 223)

This message indicates that an attempt has been made to read beyond the end of a file.

`Exists` (Error 196)

This message indicates that an attempt has been made to rename a file with a name which already exists.

`Locked` (Error 195)

This message is displayed if an attempt is made to delete or overwrite a file which has been locked (see `*ACCESS`).

`Not found` (Error 214)

This message indicates that the required file could not be found in the current (or specified) catalogue and, in the case of a `*`<file name> (or `*RUN` <file name>) command, in the current library or library Filing System.

`Open` (Error 194)

This error occurs if an attempt is made to open, delete, or overwrite a file which is already open. `*CLOSE` or `*SHUT` may be used to close files if necessary.

`Outside file` (Error 183)

This message indicates an attempt to set the file pointer beyond the end of a file opened for reading.

`Read only` (Error 193)

This message occurs if an attempt is made to write to a file opened for input.

`Syntax :` <syntax definition> (Error 220)

This is a general message which indicates that the previous command was either invalid or incomplete. The correct syntax for the command is provided.

`Too many open` (Error 192)

The Disc Filing System allows a maximum of five files to be open simultaneously. This error is displayed if an attempt is made to open a sixth file.

Error messages in numerical order

Error number		Message
183	(&B7)	Outside file
190	(&BE)	Cat full
191	(&BF)	Can't extend
192	(&C0)	Too many open
193	(&C1)	Read only
194	(&C2)	Open
195	(&C3)	Locked
196	(&C4)	Exists
197	(&C5)	Drive fault
198	(&C6)	Disc full
199	(&C7)	Disc fault
200	(&C8)	Disc changed
201	(&C9)	Disc read only
203	(&CB)	Bad option
204	(&CC)	Bad name
205	(&CD)	Bad drive
206	(&CE)	Bad dir
207	(&CF)	Bad attribute
214	(&D6)	Not found
220	(&DC)	Syntax
222	(&DE)	Channel
223	(&DF)	EOF

J.5 DFS technical information

A disc surface formatted for use with Disc Filing System consists of either 40 or 80 tracks, each divided in to 10 sectors each of 256 bytes. A given sector may therefore be specified by means of either its track and sector number or by means of its 'absolute' sector number (ie 10*<track number> + <sector number on track>). Absolute sector numbers are used throughout this section.

Files stored by the Disc Filing System are merely sequences of bytes which always begin at the start of a sector and extend for the number of (complete) consecutive sectors necessary to accommodate the data contained in the file (ie there may be a number of 'unused' bytes at the end of the last sector allocated to the file). The last 'data' byte in the file is derived from the file length stored in the catalogue entry for the file (see below) although special 'end of data' markers are sometimes used. The BASIC language, for example, uses a byte containing &FF in place of an allowable line number.

Certain items of general information and all the information necessary to access files is held in the disc catalogue which always occupies sectors 0 and 1 – storage of data for files begins at sector 2.

Note. Since track 0 is the outermost track on both 40- and 80-track discs, it is possible to access the catalogue of both types of disc in either type of drive. The same is true for any files whose content can be guaranteed to exist solely on track 0 (sector numbers 0 – 9), although this facility should be used with caution.

The disc catalogue

The format of the two sectors allocated to the disc catalogue is shown below. Note that the Disc Filing System uses 18-bit addresses (in the range 0 – &3FFFF) which are stored in two bytes (8 low order bits and 8 middle order bits) and two high order bits. These components are not necessarily stored in consecutive bytes in the catalogue.

A catalogue display produced by either `*EX` or `*INFO` will show the files in the catalogue in their correct relative positions; `*CAT` sorts the file names into order according to the ASCII codes of their constituent characters.

Sector 0

Bytes	Content
0 – 7	Leftmost eight bytes of the disc title
8 – 14	First file name
15	Directory of first file name
16 – 22	Second file name
23	Directory for second file name
.	
.	Repeated for up to 31 files
.	

The presence of the L attribute is indicated by the setting of bit 7 in the directory entry for each file:

bit 7 set	indicates that the file is locked;
bit 7 clear	indicates that the file is unlocked.

Sector 1

Bytes	Content
0 – 3	Rightmost four bytes of disc title
4	Master sequence number (see note)
5	Number of catalogue entries multiplied by 8 (used as a pointer to the next free catalogue entry)
6	(bits 0,1) number of sectors on disc (two high order bits of a 10-bit number) (bits 2,3) unused (bits 4,5) start-up option (as set by `*OPT4`) (bits 6,7) unused
7	Number of sectors on disc (eight low order bits of 10-bit number)
8	First file's load address, low order bits
9	First file's load address, middle order bits
10	First file's execution address, low order bits
11	First file's execution address, middle order bits
12	First file's length in bytes, low order bits
13	First file's length in bytes, middle order bits

14	(bits 0,1) First file's start sector (two high order bits of a 10-bit number)
	(bits 2,3) First file's load address, high order bits
	(bits 4,5) First file's length in bytes, high order bits
	(bits 6,7) First file's execution address, high order bits
15	First file's start sector (eight low order bits of a 10-bit number)
.	
.	Repeated for up to 31 files
.	

The master sequence number is a cyclic count of the number of catalogue accesses (held in binary coded decimal (BCD) format). It is set to zero when a disc is formatted and is incremented each time a file is either entered into or deleted from the catalogue.

J.6 ADFS command summary

A summary of the commands available under the Advanced Disc Filing System is given below. Commands marked 'common' under the heading 'type' are described in section G.5; those marked 'intrinsic' are specific to the Advanced Disc Filing System and are described in the next section.

Command	Function	Type
*ACCESS	Set file attributes	Intrinsic
*ADFS	Select the Advanced Disc Filing System	Intrinsic
*APPEND	Append subsequent keyboard input to a file	Common
*BACK	Make the previously selected directory current	Intrinsic
*BUILD	Send subsequent lines of keyboard input to a file	Common
*BYE	End an ADFS session	Intrinsic
*CAT	Display a directory catalogue	Common
*CDIR	Create a subordinate directory	Intrinsic
*CLOSE	Close all currently open sequential disc files	Common
*COMPACT	Reorganise free space on a disc	Intrinsic
*COPY	Copy file(s) from one directory to another	Intrinsic
*CREATE	Reserve space for a file	Common
*DELETE	Delete the name of a file	Common
*DESTROY	Delete the name(s) of a file or group of files	Intrinsic
*DIR	Set current directory	Intrinsic
*DISMOUNT	Release an ADFS drive	Intrinsic
*DUMP	Display a hex. and ASCII dump of a file	Common
*EX	Display Filing System information for files	Common
*EXEC	Take subsequent lines of input from a file	Common
*FADFS	Select the Advanced Disc Filing System	Intrinsic
*FREE	Display free space information	Intrinsic
*INFO	Display information about a file or group of files	Common
*LCAT	Display the file names in the current Library	Intrinsic
*LEX	Display Filing System information for the Library	Intrinsic
*LIB	Set current Library directory	Intrinsic
*LIST	Display the content of a file (with line numbers)	Common
*LOAD	Load file into memory	Common
*MAP	Display free space map	Intrinsic
*MOUNT	Initialise an ADFS disc	Intrinsic
*OPT0	Restore default Filing System options	Common
*OPT1	Set reporting level during file operations	Common
*OPT4	Set auto-start option	Common
*PRINT	Display a file in pure ASCII	Common

*REMOVE	Delete a file from the current catalogue	Common
*RENAME	Change the name and or directory of a file	Intrinsic
*RUN	Load and execute a machine code program	Common
*SAVE	Save a block of memory as a file	Common
*SPOOL	Send all subsequent VDU output to a file	Common
*SPOOLON	Append all subsequent VDU output to a file	Common
*TITLE	Define a directory title	Intrinsic
*TYPE	Display the content of a file (without line numbers)	Common

J.7 ADFS command descriptions

This section contains descriptions of all intrinsic Advanced Disc Filing System commands – where appropriate, the minimum abbreviation is given with the command name.

Whilst the syntax descriptions use the definitions given in section G.2 apply to all commands, the Advanced Disc Filing System allows a far more extensive use of wildcard facilities (the characters `*` and `#`) – a feature which is particularly useful for abbreviating long pathnames when referring to a file.

Note also that the symbols `@`, `&` and `^` may also be used in <object> specifications:

`@` denotes the current directory;
`^` denotes the 'parent' directory, ie the name of the directory of which the current directory is a member;
`&` is equivalent to $ (ie the root directory) and is included for compatibility with the Advanced Network Filing System (ANFS).

*ACCESS *AC.

Syntax

`*ACCESS` <object> `[L][W][R][E]`

Purpose

To set access restrictions ('attributes') for a file (or group of files).

Description

`*ACCESS` sets the file attributes according to the presence or absence of one or more of the optional parameters following the <object> specification.

`L` locks the objects so that they may not be deleted, written to or overwritten (for example, if a locked file is loaded, it may not be saved again with the same name.)

Both files and directories may be locked. Note, however, that locking does not prevent directories or files from being destroyed if the disc is reformatted with the `AFORM` or `BACKUP` utilities (see section J.9).

`W` sets 'write access' (for files only). This must be set for writing and updating to be allowed.

`R` sets 'read access' (for files only). This must be set for reading and loading to be allowed.

`E` sets 'execute only access' (for files only) and removes W and R attributes if set.

This facility is provided for protecting machine code program files and when set, prevents the use of `*LOAD` and inhibits the catalogue information on the file for `*EX` and `*INFO` commands (and all OSFILE calls except 6 (Delete)). The only commands which may specify a file with execute only access are `*RUN`, `*`<file name>, `*DELETE`, `*REMOVE`, `*DESTROY`.

`*ACCESS` may subsequently be used to set (or remove) the L attribute only.

There is a further attribute – D which is set if the object is a directory (see `*CDIR` below).

The default attributes (i.e. those which are assigned when an object is first created) are:

Files	:	WR
Directories	:	DLR

*ADFS *A.

Syntax

`*ADFS`

Purpose

To select the Advanced Disc Filing System (see also `*FADFS`, below).

Description

`*ADFS` selects the Advanced Disc Filing System and makes it the current Filing System.

On the first use of *ADFS since a power-on (or if the Advanced Disc Filing System is set as the default Filing System), the action taken depends upon the computer's configuration and the configuration status:

- For systems equipped with a Winchester disc drive only, *ADFS will select it (drive 0);
- For systems equipped with floppy drive(s) only, *ADFS will select floppy drive 0;
- For systems equipped with both a Winchester disc drive and floppy drive(s), *ADFS will select the drive specified by the setting of *CONFIGURE HARD/FLOPPY:

  *CONFIGURE HARD will cause *ADFS to select the Winchester drive (0);
  *CONFIGURE FLOPPY will cause *ADFS to select the floppy drive (4);

In addition, the setting of *CONFIGURE DIR/NODIR determines whether the root directory is to be loaded from the selected drive:

*CONFIGURE DIR	causes ADFS to access the specified drive and load the root directory into the ADFS workspace, making it the current directory. The library directory remains 'unset' unless the root directory so loaded contains a directory whose name begins with the characters LIB (or Lib or lib etc.), in which case it becomes the current library directory.
*CONFIGURE NODIR	prevents ADFS from accessing the disc. Both the current directory and the library directory remain 'unset' and must be set explicitly by using the *DIR, *LIB or *MOUNT commands (as described below).

Subsequent uses of *ADFS will select the Advanced Disc Filing System with the current and library directories set to those last in use.

*BACK

Syntax

*BACK

Purpose

To make the previously selected directory current.

Description

In addition to the current directory, the Advanced Disc Filing System maintains a record of the last directory selected and this may be made current by the use of *BACK. A common use is for switching between two frequently-used directories.

*BYE

Syntax

*BYE

Purpose

To end an ADFS session permanently.

Description

*BYE is similar in effect to *CLOSE in that it ensures that any currently open sequential files are closed with any data remaining in the associated buffer written to the file. In addition, however, *BYE moves the read/write heads to a 'transit position' – this operation is vital if a Winchester disc unit is to be moved.

*CDIR *CD.

Syntax

*CDIR <pathname>

Purpose

To create a new directory.

Description

*CDIR creates a new, empty directory with the specified name in the position in the directory hierarchy specified by <pathname>.

The new directory is assigned DLR attributes (see *ACCESS) and a master sequence number of zero (see section J.10).

The directory name is also allocated as the directory title (see *TITLE).

*COMPACT *COM.

Syntax

`*COMPACT` [<start page number> <number of pages>]

Purpose

To reorganise the free space on a disc so that it occupies larger, contiguous blocks.

Description

Attempting to save a program or file on to a disc may produce the messages `Map full` or `Compaction required`, which indicate that the free space available on the disc is insufficiently large to accommodate the new file. It may be that there is enough space in total, but that it is split into several small sections created by deleting files.

`*COMPACT` reorganises the free space map, gathering free areas into larger blocks.

The parameters allow a designated area of RAM to be used as a buffer during the compaction process – they are optional in non-shadow modes only; `*COMPACT` uses the current screen memory (and produces some bizarre, but inconsequential screen effects) if they are omitted.

<start page number>	is a two-digit hexadecimal value indicating the page at which the buffer is to start (ie 30 indicates page &30 – absolute address &3000 in memory);
<number of pages>	is a two-digit hexadecimal value indicating the number of pages to be used (ie 40 indicates a buffer length of &40 pages (16K bytes)).

Care must be taken to ensure that the absolute address of the end of the buffer is in the IO processor's RAM (ie < &8000).

The speed at which `*COMPACT` operates depends upon the size of the buffer to be used – optimal performance is achieved in any of the Shadow modes using the command:

`*COMPACT 0E 72` (the entire user RAM in the IO processor)

Clearly, the content of a user specified buffer will be overwritten during the compaction process.

*COPY

Syntax

`*COPY` <object> [<drive specification>]<pathname>

Purpose

To copy files from one directory to another.

Description

`*COPY` copies all files matching the specification of <object> to the directory specified as the second parameter. If necessary, files may be copied, say, from a Winchester disc to floppy disc by including appropriate <drive specification> parameters. The file(s) are added to the catalogue for the specified directory and it is therefore possible to encounter `Dir full` or `Disc full` errors.

`*COPY` uses some or all of the memory between OSHWM and the bottom of the current screen memory as a workspace during the copying process – optimal performance is achieved in either Mode 7 or any of the Shadow modes.

*DESTROY *DES.

Syntax

`*DESTROY` <object>

Purpose

To delete specified files in a single action.

Description

The common command `*DELETE` is used to delete a single file; `*DESTROY` performs an identical function if a unique <object> is specified but the use of the wildcard facilities allows a group of files to be deleted in one operation.

`*DESTROY` first displays a list of the (unlocked) files matching the specification of <object> and then displays the prompt:

```
Destroy ?
```

If the reply is `YES` (followed by RETURN), the names of the objects are deleted from the current directory catalogue. Any other reply aborts the command.

Directory names matching <object> may only be deleted if they are unlocked and empty.

*DIR

Syntax

`*DIR` [<drive specification>][<pathname>]

Purpose

To change the current directory.

Description

By default, the current directory is set to `$` (the root directory) – this is restored by the use of a `*DIR` command with no parameters.

Where a parameter is given, the current directory specification is saved (for use by `*BACK`) and the directory specified is made current.

Note that a command such as:

```
*DIR :4.MARTYN
```

is equivalent in effect to the commands:

```
*MOUNT 4                    (see below)
*DIR MARTYN
```

*DISMOUNT *DISM.

Syntax

`*DISMOUNT` [<drive number>]

Purpose

To release an ADFS drive (normally prior to changing discs).

Description

`*DISMOUNT` closes all currently open sequential files on either the current or specified drive. In addition, if the current directory is on the dismounted drive, the Filing System is put into a state such that the current directory and the library are 'unset' . In this state, a `*CAT` command will display the content of the root directory for the drive, but any attempt to access a file will result in output of the message:

```
No directory
```

A new disc may be initialised using either `*ADFS`, `*MOUNT` (see below) or by setting the current directory by means of `*DIR`.

*FADFS *FA.

Syntax

`*FADFS`

Purpose

To select the Advanced Disc Filing System (see also `*ADFS` above).

Description

`*FADFS` selects the Advanced Disc Filing System but without any access to a disc drive – regardless of the setting of `*CONFIGURE DIR/NODIR`. Both the current and library directories remain 'unset'.

*FREE *FR.

Syntax

`*FREE`

Purpose

To display free space available on the current drive.

Description

`*FREE` displays the total amount of free space left and the amount of space used on the current drive. The format of the display is:

```
<ssssss>  Sectors =  <bbbbbbbbb>  Bytes free
<ssssss>  Sectors =  <bbbbbbbbb>  Bytes used
```

where:

<ssssss> is the number of sectors free/used (in hexadecimal);
<bbbbbbbbb> is the number of bytes free/used (in decimal).

Note that the values for the free space are totals and do not take the fragmentation of the free space into account – see `*MAP`.

*LCAT *LC.

Syntax

`*LCAT`

Purpose

To display the catalogue for the current library directory.

Description

*LCAT is equivalent to *CAT with the library directory specified as its parameter.

*LEX

Syntax

*LEX

Purpose

To display Filing System information for the library directory.

Description

*LEX provides exactly the same facility as *EX, but for the library directory and without the need to make the library directory current.

*LIB

Syntax

*LIB <pathname>

Purpose

To set the library directory.

Description

By default, the library directory is either 'unset' (if ADFS is entered using *FADFS and the current directory is specified using *DIR or if the current drive was initialised using *MOUNT), or set to the first directory in the root whose name starts with the characters LIB (if ADFS is entered using *ADFS).

*LIB sets the library directory to that specified.

Note that unlike *DIR, *LIB does not affect either the current drive or the current directory, it specifies the directory to be examined in the object of a *<file name> command cannot be located in the current directory. The search may be extended to include the current and library directories of the Library Filing System if required (see section G.2).

*MAP

Syntax

*MAP

Purpose

To display a map of the free space available on a disc.

Description

Whereas *FREE gives an indication of the total amount of free space, *MAP displays a map of the distribution of the free space, in terms of start sector numbers and lengths, as shown below. If the <drive number> is omitted, the current drive is assumed.

```
Address   :  Length
<aaaaaa>  :  <llllll>
             .
             .
<aaaaaa>  :  <llllll>
```

where:

<aaaaaa>	is the sector address of a free space
<llllll>	is the length of the space in sectors (displayed in hexadecimal)

The number of entries in the free space map displayed by *MAP is a good guide to the likely need for compaction. The message:

```
Compaction required
```

is output if the number of entries in the free space map reaches 80 although it is recommended that a disc be compacted if the free space map contains more than 60 entries.

*MOUNT *MOU.

Syntax

*MOUNT [<drive number>]

Purpose

To initialise an ADFS disc.

Description

*MOUNT initialises an ADFS disc (i.e. it reads the free space map and the root directory catalogue into memory and makes the specified drive current; if no <drive number> is specified, *MOUNT re-mounts the current drive).

If the library is currently set on the drive specified, it is unset by *MOUNT and must be set using a *LIB command. The library is unaffected if it is on a different drive.

*RENAME *REN.

Syntax

`*RENAME` <object> <object>

Purpose

To change the name (and/or directory) of a file.

Description

`*RENAME` changes the catalogue information relating to a specified object from that specified by the first parameter to that specified by the second. If either or both <directory specification>s are omitted, the current directory is assumed.

Note that it is not possible to rename a locked object and that the object defined by the second parameter must not already exist. If the object is a directory, all files and subordinate directories remain unchanged but will be accessible only via their new pathname.

The syntax of `*RENAME` does not allow the drive number to be changed as this information is not held in the catalogue information for either files or directories.

*TITLE *TIT.

Syntax

`*TITLE` <title string>

Purpose

To specify, or change a directory title.

Description

By default, each directory is assigned its directory name as a title. `*TITLE` enables the title string associated with each directory to be changed – the directory title has no significance to the Filing System.

<title string> is a sequence of characters which will be written to title field of the current directory and which will subsequently be displayed for every `*CAT` command.

The string is terminated by the carriage return character used to terminate the command and may contain spaces if required. The title stored in the directory is either truncated or space filled (at the right) to 19 characters.

J.8 ADFS error messages

The messages given below may be displayed by the Advanced Disc Filing System.

`Aborted` (Error 146)

This message is displayed if a response other than `YES` (or `yes`, `Yes` etc.) is given to the prompt:

```
Destroy ?
```

after a `*DESTROY` command.

`Access violation` (Error 189)

This message indicates that an attempt has been made to either:

– read or load a file which does not have an R attribute;
– read or load a file with an E attribute;
– write or save to a file which does not have a W attribute.

`Can't - file open` (Error 194)

This error occurs if an attempt is made to open, delete or overwrite a file which is already open. `*CLOSE` or a `*SHUT` may be used to close the file if necessary.

`Already exists` (Error 196)

This message is produced if a `*CDIR` or a `*RENAME` command attempts to create an object which already exists. Note that `*SAVE`, `*BUILD` etc (and any language implementation of similar commands will overwrite an existing file unless its L attribute is set.

`Bad sum` (Error 170)

This is a 'fatal' error indicating a corruption in RAM which prevents ADFS from accessing a file. The system must be re-started by a hard break.

`Bad FS map` (Error 169)

This is a 'fatal' error indicating either corruption in RAM (which may normally be cleared by a hard break) or corruption of sectors 0 and 1 on the current disc – see section J.10).

`Bad name` (Error 204)

This message indicates the use of a file name which is invalid, for example, too long, missing, or containing invalid characters.

`Bad opt` (Error 203)

This message indicates the use of invalid parameters in a `*OPT` command. Only

`*OPT1` and `*OPT4` are recognised by the Advanced Disc Filing System – see section G.5

`Bad compact` (Error 148)

This message indicates the use of invalid parameters with a `*COMPACT` command or the omission of the parameters if `*COMPACT` is used in a shadow mode.

`Bad rename` (Error 176)

This message indicates that a `*RENAME` command referring to a directory would produce an illegal directory structure if executed – normally by attempting to allow a directory entry to refer to itself.

`Broken directory` (Error 168)

This message is produced if the Filing System detects corruption in the format of a directory on the disc. The implication is that the disc is in an inconsistent state and it should be re-formatted if possible.

`Can't delete CSD` (Error 150)

This message indicates an attempt to delete the current directory.

`Can't delete Library` (Error 151)

This message indicates an attempt to delete the current library directory.

`Channel on channel` <nn> (Error 222)

This message indicates an attempt to access a file which has not been opened for byte access and for which, therefore, no file handle has been allocated by the Filing System.

`Compaction required` (Error 152)

This message indicates that the free space on the disc has become too fragmented ie there are 80 entries in the free space map. The disc must be compacted using `*COMPACT`.

The message:

`Compaction recommended`

(which is NOT an error) is displayed after every filing system command once the free space map becomes approximately three-quarters full (approx. 60 entries).

`Data lost on channel` <nn> (Error 202)

This message indicates that a disc error of some sort occurred during a file read or write operation. <nn> is the channel number in hexadecimal.

`Dir full` (Error 179)

This message indicates an attempt to create a new entry in a directory which already contains 47 entries.

`Dir not empty` (Error 180)

This message indicates an attempt to delete a directory which still contains objects.

`Disc changed` (Error 208)

This message indicates that the Advanced Disc Filing System has detected a discrepancy between the current disc and catalogue information held in RAM.

`Disc error <nn> at : <d>/<ss>` (Error 199)

This message indicates that the disc controller detected a fault during the last operation:

<nn> is the fault number (in hexadecimal);
<d> is the drive number;
<ss> is the sector number (in hexadecimal).

See section J.11 for a list of disc fault numbers.

`Disc full` (Error 198)

This message indicates that there is insufficient free space to allow completion of the current operation. The problem may sometimes be overcome by compacting the disc (ie amalgamating disjoint blocks of free space into larger blocks).

`Disc protected` (Error 201)

This message indicates an attempt to write to a (floppy) disc which is protected by means of a write protect tab.

`Drive not ready` (Error 205)

The occurrence of this message indicates that the disc controller is unable to access the disc drive – often caused by attempting to access a Winchester disc before it has reached its operational speed.

`EOF on channel <nn>` (Error 223)

This message indicates an attempt to read beyond the end of a file.

`Locked` (Error 195)

This message indicates an attempt to delete, overwrite or rename a file for which the L attribute is set.

`Map full` (Error 153)

In contrast to `Disc full` (above), this message indicates that although there may be free space on the disc, there is no more space available to extend the free space map. The disc must be compacted.

`No directory` (Error 169)

This message is produced if an attempt is made to access an object when there is no current directory – typically immediately after a `*FADFS` command.

`Not found` (Error 214)

This message indicates that the required object could not be found, ie:

– a directory name not in the current directory;
– a file name not in the current directory and, in the case of a *<file name> command, not in the current library directory or on the library Filing System.

`Not open for update on channel` <nn> (Error 193)

This message indicates an attempt to write to a file opened for input only. <nn> is the channel number (in hexadecimal).

`Outside file on channel` <nn> (Error 183)

This message indicates an attempt to set the file pointer beyond the end of a file opened for reading.

`Too many open files` (Error 192)

The Advanced Disc Filing System allows a maximum of ten files to be open simultaneously. This message indicates an attempt to open an eleventh file.

`Wild cards` (Error 253)

This message indicates the illegal use of the wildcard characters in parameters to commands which require explicit references.

`No!` (Error 147)

This message indicates an attempt to `*RUN` a file whose load address is &FFFFFFFF.

Error messages in numerical order

Error	number	Message
146	(&92)	Aborted
147	(&93)	No!
148	(&94)	Bad compact
150	(&96)	Can't delete CSD
151	(&97)	Can't delete library
152	(&98)	Compaction required
153	(&99)	Map full
168	(&A8)	Broken Directory
169	(&A9)	Bad FS map *or* No directory
170	(&AA)	Bad sum
176	(&B0)	Bad rename
179	(&B3)	Dir full
180	(&B4)	Dir not empty
183	(&B7)	Outside file
189	(&BD)	Access violation
192	(&C0)	Too many open files
193	(&C1)	Not open for update
194	(&C2)	Can't - file open
195	(&C3)	Locked
196	(&C4)	Already exists
198	(&C6)	Disc full
199	(&C7)	Disc error
200	(&C8)	Disc changed
201	(&C9)	Disc protected
202	(&CA)	Data lost
203	(&CB)	Bad opt
204	(&CC)	Bad name
205	(&CD)	Drive not ready
214	(&D6)	Not found
222	(&DE)	Channel
223	(&DF)	EOF
253	(&FD)	Wild cards

J.9 ADFS utilities

A number of utility programs for use with the Advanced Disc Filing System are provided on the Welcome disc supplied with the computer. The means of executing each utility is described in the accompanying Welcome Guide.

Utility	Function
AFORM	Format a floppy disc for use with ADFS
BACKUP	Copy the content of one disc to another
CATALL	Display a full catalogue for a disc
COPYFILES	Copy files to another Filing System
DIRCOPY	Copy all or part of the directory structure
EXALL	Display Filing System information for all files on a disc
HARDERROR	Instruct the Filing System to ignore known disc faults
RECOVER	Recover a file deleted by accident
VERIFY	Verify the formatting on a disc

AFORM

Purpose

To format a floppy disc for use with the Advanced Disc Filing System.

Description

In order to be able to format a disc, AFORM requires the following information:

– the drive number

  which will be 0 for a single drive and either 0 or 1 for a dual drive. Note that the floppy drive numbers are 4 (and 5) if the system is equipped with both floppy disc drive(s) and a Winchester drive.

– the 'size' of the disc

  three options are available:

  – single-sided 40-track specified as `S` (Small) or 40;
  – single-sided 80-track specified as `M` (Medium) or 80;
  – double-sided 80-track specified as `L` (Large) or 160.

Either or both parameters may be specified explicitly if the program is executed by means of the command `*AFORM` and the user will be prompted for any parameters which are not supplied. Prompts will be issued for both parameters if the program is executed in any other way.

Pressing RETURN in response to a prompt will produce a summary of the available options.

Once the parameters have been specified, AFORM displays a confirmatory message to which the user must reply `YES` (or `yes` or `Yes`) if he wishes to format the disc. Any other response will abort the format.

ENSURE THAT THE DISC TO BE FORMATTED IS IN THE CORRECT DRIVE.

A running count of the track numbers is displayed as the formatting is carried out and the formatted disc is then verified.

On completion, the formatted disc will contain an empty root directory into which files and new directories may be entered as required.

BACKUP

Purpose

To copy the content of one disc to another.

Description

BACKUP performs a sector by sector copy from one disc to another, thereby creating an exact duplicate.

Two parameters are required, the first being the 'source' drive (ie the drive containing the disc from which the copy is to be made); the second being the (optional) 'destination' drive (ie the drive containing the disc to which the copy is to be directed).

ALL DATA ON THE DESTINATION DISC WILL BE OVERWRITTEN.

Either or both parameters may be specified explicitly if the program is executed by means of a `*BACKUP` command; the user will be prompted for both parameters if the program is executed in any other way.

If the destination drive is the same as the source drive (or if the destination drive parameter is omitted) the backup will be performed in one drive only, the user being prompted to change discs when necessary.

CATALL

Purpose

To display catalogue information for a complete disc.

Description

CATALL is a BASIC program which will first prompt the user for the drive to be catalogued.

The display produced is in the same format as for a `*CAT` command but it operates on each directory in turn, starting with the root and proceeding down through the hierarchy.

CATALL can produce a great deal of output – the display may be temporarily 'frozen' using simultaneous depression of SHIFT and CTRL. Alternatively, paged mode (CTRL+N) or a printer may be selected before CATALL is run.

COPYFILES

Purpose

To copy files from one filing system to another.

Description

COPYFILES is a BASIC program which will prompt the user for the information it requires to be able to carry out the copying function. Depression of RETURN in reply to a prompt will display the possible responses.

The prompts cover:

- the source Filing System;
- the source drive number (if appropriate);
- the source directory (if appropriate);
- the destination Filing System;
- the destination drive number (if appropriate);
- the destination directory (if appropriate);
- the copying mode;
- details of the source and destination file names.

The copying mode may be one of:

- Single file mode

  The source and destination files may be anywhere in the filing system structure – ie the names may include a pathname if appropriate. The source name will be used as the destination name if the latter is not specified.

– Multiple mode

  A group of files must be specified by means of the wildcard characters `#` and `*`; a `Y/N` prompt is issued for all of the source files (in the source directory) matching the definition; only those files for which `Y` is the response are copied to the destination.

– List mode

  copies all files matching the wildcard specification from the source to the destination without the prompts issued in multiple mode.

DIRCOPY

Purpose

To copy all or part of the directory structure to a new area, including copying to another disc.

Description

DIRCOPY is a BASIC program which will prompt the user for the information necessary to perform the copy. These are:

– source drive number;
– destination drive number;
– source 'root' ie the 'highest' directory in the hierarchy (specified as a full pathname from the absolute root directory (`$`);
– destination 'root' ie the highest directory in the hierarchy (specified as for the source);
– the names of any directories to be excluded from the copy; reference to a directory in response to this prompt will cause the specified directory (and any subordinate directories) to be excluded.

With this information, DIRCOPY displays a catalogue of the source root directory and issues a further prompt for the first file to be copied. Depression of RETURN starts the copy from the first file in the directory.

DIRCOPY takes file attributes into account both on reading from the source and on writing to the destination – source files without read access will be ignored and existing, locked destination files will not be overwritten (although a prompt allows this attribute to be ignored if required).

EXALL

Purpose

To display Filing System information for all files on a disc.

Description

EXALL is a BASIC program which will prompt the user for the number of the drive from which information is to be displayed.

The display produced is in the same format as for a `*EX` command but it operates on each directory in turn, starting with the root and proceeding down through the hierarchy.

EXALL can produce a great deal of output – the display may be temporarily 'frozen' using simultaneous depression of SHIFT and CTRL. Alternatively, paged mode (CTRL+N) or a printer may be selected before EXALL is run.

HARDERROR

Purpose

To instruct the Filing System to ignore known disc faults.

Description

HARDERROR is intended for use in the rare instance of a permanent disc error on a floppy disc and where re-formatting would destroy useful data.

HARDERROR operates only on the Free Space Map maintained on the disc and the 'recovery' is achieved by deleting the free space entry which references the faulty track/sector. If the fault lies within an existing object, it must first be deleted so that the area is entered into the Free Space Map. This will entail deletion of all constituent entries if the object is a directory.

Once HARDERROR has been used on a disc, it must NOT be compacted using `*COMPACT`.

RECOVER

Purpose

To recover a file deleted by accident.

Description

The `*DELETE` and `*DESTROY` commands do not remove a file from the disc in its entirety; only the name is removed from its parent directory. Files so deleted may be recovered provided that the space that the file occupied has not yet been overwritten.

Before running RECOVER, two directories:

```
$.LostFiles.A
$.LostFiles.B
```

must exist on the disc. (It is a good idea to create these directories BEFORE any files are deleted since creating them afterwards could easily overwrite the file you are trying to recover!)

RECOVER uses these two directories to build 'files' from the content of the free areas of the disc (some of which will contain the sectors of the file to be recovered). The user is then able to examine the content of these files and to save sectors as required.

Full instructions are provided while the program is running.

VERIFY

Purpose

To verify the formatting on a floppy disc.

Description

Discs are verified automatically when they are formatted by the AFORM utility but it is sometimes necessary to check the formatting on a disc at other times, particularly if disc errors or other problems are encountered.

VERIFY verifies a specific drive number; this may be specified as a parameter if the program is executed by means of a `*VERIFY` command; the user will be prompted for the drive number if it is omitted or if the program is executed by any other means.

VERIFY checks each sector to determine if it is readable.

J.10 ADFS technical information

A disc formatted for use with the Advanced Disc Filing System will be either:

– single-sided

  in which case it will contain either 40 or 80 tracks, each divided into 16 sectors containing 256 bytes.

– double-sided

  in which case it will contain 2 x 80 = 160 tracks, each divided into 16 sectors containing 256 bytes. (The two surfaces are considered to be a single entity).

A given sector may therefore be identified by means of either its track and sector number or by means of its 'absolute' sector number (ie 16*<track number>+<sector number on track>. Absolute sector numbers are used throughout this section.

Files stored by the Advanced Disc Filing System are merely sequences of bytes which always begin at the start of a sector and extend for the number of (complete) sectors necessary to accommodate the data contained on the file (ie there may be a number of 'unused' bytes at the end of the last sector allocated to the file). The last 'data' byte in the file is derived from the file length stored in the catalogue entry for the file (see below).

Unlike the Disc Filing System (in which the areas of free space are derived from the catalogue entries for each file), the Advanced Disc Filing System maintains a map of the free space (and other information) in sectors 0 and 1 of each disc.

Sectors 2 to 6 inclusive always contain the information relating to the root directory and the remaining sectors contain either information relating to files and subordinate directories or the actual content of files.

Note. Since track 0 is the outermost track on both 40- and 80-track discs, it is possible to access both the Free Space Map and the information relating to the root directory of both types of disc in either type of drive. The same is true for any files or directories which can be guaranteed to exist solely on track 0 (sector numbers 0 – 15), although this facility should be used with caution.

A reference to an absolute sector number is always represented by a 3-byte value.

The Free Space Map

The Free Space Map is stored in sectors 0 and 1 on each drive. The format being:

Sector 0

Bytes	Content
0 – 2	Start sector of first free space
3 – 5	Start sector of second free space
6 – 8	Start sector of third free space
.	
.	Repeated for 82 free space entries
.	
246	Reserved
247	Reserved
248	Reserved
249	Reserved
250	Reserved
251	Reserved
252	LSB of total number of sectors on disc
253	.
254	MSB of total number of sectors on disc
255	Checksum on free space map, sector 0

Sector 1

Bytes	Content
0 – 2	Length of first free space (in sectors)
3 – 5	Length of second free space
6 – 8	Length of third free space
.	
.	Repeated for 82 free space entries
.	
246	Reserved
247	Reserved
248	Reserved
249	Reserved
250	Reserved
251 – 252	Disc identifier
253	Boot option number (as set by `*OPT4`)
254	Pointer to end of free space list
255	Checksum on free space map, sector 1

Directory information

A directory consists of five contiguous sectors on the disc – the root directory is always located on track 0 (sectors 2 – 6). The information relating to the current directory is always resident in the area of RAM reserved for use by the MOS/Filing Systems.

Note that whereas the Disc Filing System uses 18-bit addresses, all addresses held by the Advanced Disc Filing System are 32 bits (4 bytes) in length.

The format for each directory is shown below.

Bytes	Content
0	Directory Master Sequence Number (in binary coded decimal)
1 – 4	Fixed string identifying the sector as the start of a directory
5 – 14	Name and access string for first directory entry (see note)
15 – 18	Load address for first directory entry
19 – 22	Execution address for first directory entry
23 – 26	Length of first directory entry in bytes
27 – 29	Start sector for first directory entry
30	Sequence number for first directory entry (see note)
31 – 40	Name and access string for second directory entry
41 – 44	Load address for second directory entry
45 – 48	Execution address for second directory entry
49 – 52	Length of second directory entry in bytes
53 – 55	Start sector for second directory entry
56	Sequence number for second directory entry
.	
.	Repeated for 47 directory entries
.	
1227	0
1228 – 1237	Directory name/access string
1238 – 1240	Start sector of parent directory
1241 – 1259	Directory title
1260 – 1273	Reserved
1274	Directory Master Sequence Number (in binary coded decimal)
1275 – 1278	Fixed string identifying a directory
1279	0

Notes

A directory can take a maximum of 47 entries – if there are less than 47, this is indicated by the entry after the last file having a name string starting with &00. Directory entries are held in alphabetical order.

File and directory attributes are stored in the top bit of the first four bytes of the name/access string string:

1st byte bit 7 set/clear indicates R attribute set/not set
2nd byte bit 7 set/clear indicates W attribute set/not set
3rd byte bit 7 set/clear indicates L attribute set/not set
4th byte bit 7 set indicates that the entry is a directory
bit 7 clear indicates that the entry is a file

If a string is shorter than the space reserved for it, the string ends with a &0D and the remaining bytes are not significant.

Each directory has a two-byte master sequence number, held in binary coded decimal format. This value is set to zero when the directory is created.

The master sequence number is incremented every time a change is made to the content of the directory catalogue and the new value is assigned to the sequence number associated with the new/changed directory entry. The sequence number for each directory entry (as displayed by `*CAT`) may therefore be used to assess the 'age' of each file.

J.11 Using DFS and ADFS from assembly language

The routines described below are implemented by both the Disc Filing System and the Advanced Disc Filing System unless indicated to the contrary.

OSFIND — Open or close a file for byte access

Call address &FFCE
Indirected via &21C (FINDV)

On entry : A defines the action to be taken.

X and/or Y contain values depending upon the action specified.

Actions specified by A :

A = 0 (&00) indicates that a file is to be closed.

Y may contain either the <file handle> of the file to be closed or zero, which indicates that all currently open sequential files associated with the current Filing System are to be closed.

A <file handle> is allocated by the Filing System when a file is opened.

A = 64 (&40) indicates that a file is to be opened for input.

X and Y point to the location in memory (X=LSB, Y=MSB) containing the first character of the file name (which must be terminated by a carriage return character).

The named file must exist.

A = 128 (&80) indicates that a file is to be opened for output.

X and Y point to the file name as described above.

If the named file exists, it will be opened and its file pointer set to the start of the file; If the file does not exist, a new file is created with a default length depending upon the Filing System:

DFS: 16K (&4000) bytes
ADFS: 64K (&10000) bytes

A = 192 (&C0) indicates that a file is to be opened for input and output (random access).

X and Y point to the file name as described above.

The named file must exist.

On exit : X and Y are preserved

A is preserved if zero on entry, otherwise A contains the file handle allocated to the file. A value of zero indicates that the Filing System was unable to open the file.

C, N, D and V are undefined.

The interrupt state is preserved but may be enabled during the operation.

OSGBPB — Read or write a group of bytes

Call address &FFD1
Indirected via &21A (GBPBV)

On entry : A defines the action to be taken

X = <LSB of parameter block address>
Y = <MSB of parameter block address>

Parameter block size : 13

Parameter block format : XY = <file handle>
XY+1 = <LSB of pointer to data in memory>
XY+2 .
XY+3 .
XY+4 = <MSB of pointer to data in memory>
XY+5 = <LSB of number of bytes to transfer>
XY+6 .
XY+7 .
XY+8 = <MSB of number of bytes to transfer>
XY+9 = <LSB of sequential pointer value>
XY+10 .
XY+11 .
XY+12 = <MSB of sequential pointer value>

Not all sections of the parameter block are used by all actions.

Actions specified by A :

A = 1 (&01) Write bytes to file

The number of bytes specified in XY+5 to XY+8, starting at the address specified in XY+1 to XY+4 are written to the file using the sequential pointer value specified in XY+9 to XY+12.

A = 2 (&02) Append bytes to file

The number of bytes specified in XY+5 to XY+8, starting at the address specified in XY+1 to XY+4 are appended to the specified file (ie the bytes are written to the file starting at the current value of the file pointer, which is incremented for each byte transferred).

A = 3 (&03) Read bytes from a specified position in a file

The number of bytes specified in XY+5 to XY+8 are read from the file starting at the pointer value specified in XY+9 to XY+12.

Bytes read are placed in memory locations starting at the location contained in XY+1 to XY+4.

A = 4 (&04) Read bytes from the current position in the file

The number of bytes specified in XY+5 to XY+8 are read from the file starting at the current value of the file pointer (ie the value in XY+9 to XY+12 is ignored). Bytes read are placed in memory locations starting at the address specified in XY+1 to XY+4.

A = 5 (&05) Read title, option and drive

The current title, option and drive number are returned in the area of memory specified in XY+1 to XY+4.

Under DFS, the title is that relating to the disc in the current drive; under ADFS, the title is that of the current directory.

The format of the returned data is:

length of title	(1 byte)
title string in ASCII	(length as specified)
start option	(1 byte)
drive number	(1 byte)

The contents of XY and XY+5 to XY+13 are ignored.

A = 6 (&06) Read current drive and directory name

The current drive and the name (rather than the title) of the current directory are returned in the area of memory specified in XY+1 to XY+4

The format of the returned data is:

length of drive number	(1 byte)
drive number in ASCII	(see note)
length of directory name	(1 byte)
directory name in ASCII	(length as specified)

The drive number will always be one byte in length for DFS and ADFS – this call may return a string of more than one byte for other Filing Systems

The contents of XY and XY+5 to XY+13 are ignored.

A = 7 (&07) Read current library drive and name

As for A = 6 but with relation to the current library.

A = 8 (&08) Read file names from the current directory

The content of XY+5 to XY+8 is treated as the number of filenames to transfer; XY +9 to XY+13 contain a pointer to the first name to be transferred (i.e. if it is zero, the search will begin with the first file).

The disc Master Sequence Number is returned in XY and the file names are returned in locations specified by XY+1 to XY+4. The format of the returned data is:

length of first file name	(1 byte)
first file name in ASCII	(length as specified)
length of second file name	(1 byte)
second file name in ASCII	(length as specified)

.
. Repeated as specified by XY+5 to XY+8
.

On exit : A, X and Y are preserved.

N, V and Z are undefined

C = 0 indicates that the transfer was completed

C = 1 indicates that the transfer was incomplete for some reason.

The values in the parameter block are updated to reflect the position after the transfer (i.e. XY+1 to XY+4 contain the address of the byte after the last byte transferred to or from the file and XY+5 to XY+8 contain the number of bytes remaining to be transferred – zero if C = 0).

The interrupt state is preserved, but may be enabled during the call.

OSBPUT — Write a byte to an open file

Call address &FFD4
Indirected via &218 (BPUTV)

On entry : Y = <file handle>

A = <byte to be written>

The byte is written to the specified file using the current value of the file pointer. The pointer is incremented for each byte written.

On exit : A, X and Y are preserved

C, N, V and Z are undefined

The interrupt state is preserved but may be enabled during the call.

OSBGET — Read a single byte from an open file

Call address &FFD7
Indirected via &216 (BGETV)

On entry : Y = <file handle>

The byte at the current file pointer position is returned in A. The file pointer is incremented for each byte read.

On exit : A = <byte read>

X and Y are preserved

N, V and Z are undefined

C = 0 indicates that the transfer was completed
C = 1 indicates that end of file was encountered and that the value in A should be discarded.

The interrupt status is preserved but may be enabled during the call.

OSARGS Read filing system information

Call address &FFDA
Indirected via &214 (ARGSV)

On entry : A specifies the action to be taken.

X points to a four-byte area in page zero
Y contains either a file handle or zero

Actions specified by A :

A = 0 (&00)	Y = 0	Return Filing System number in A.
	Y = <file handle>	Return sequential pointer for file in locations specified by X.
A = 1 (&01)	Y = 0	Return address of remainder of the last command line in locations specified by X.
	Y = <file handle>	Write sequential pointer for file from locations specified by X.

A = 2 (&02)	Y = <file handle>	Return length of file (in bytes) in locations specified by X.
A = 255 (&FF)	Y = 0	Ensure any buffered data has been written to all files.
	Y = <file handle>	Ensure any buffered data has been written to the specified file.

On exit : X and Y are preserved

A is preserved except when A=0 and Y=0 on entry, in which case it contains the Filing System number (see section G.1).

The interrupt status is preserved although interrupts may be enabled during the call.

Note

Addresses returned in the four-byte block specified by the content of X ALWAYS point to the I/O processor and should therefore be read using OSWORD 5.

OSFILE — Load or save a complete file

Call address &FFDD
Indirected via &212 (FILEV)

On entry : A specifies the action to be performed

X = <LSB of parameter block address>
Y = <MSB of parameter block address>

Parameter block size : 18

Parameter block format : XY = <LSB of address of file name>
XY+1 = <MSB of address of file name>
XY+2 = <LSB of load address for file>
XY+3 .
XY+4 .
XY+5 = <MSB of load address for file>
XY+6 = <LSB of execution address for file>
XY+7 .
XY+8 .
XY+9 = <MSB of execution address for file>
XY+10 = LSB of either <start address> or <length>
XY+11 .
XY+12 .
XY+13 = MSB of either <start address> or <length>
XY+14 = LSB of either <end address> or <file attributes>
XY+15 .
XY+16 .
XY+17 = MSB of either <end address> or <file attributes>

The file name pointed to by XY and XY+1 must terminate with a carriage return.

Actions specified by A :

A = 0 (&00) Save a block of memory.

On entry, XY+10 to XY+13 contain the start address of the data in memory; XY+14 to XY+17 contain the end address.

On exit, XY+10 to XY+13 are replaced by the length of the file and XY+14 to XY+17 are replaced by the file attributes assigned by the Filing System (see below).

A = 1 (&01) Write catalogue information for the named file.

The load address, execution address and file attributes from the parameter block are written to the named file's catalogue entry.

A = 2 (&02) Write load address (only) for the named file.

A = 3 (&03) Write execution address (only) for the named file.

A = 4 (&04) Write attributes (only) for the named file.

A = 5 (&05) Read catalogue information for the named file.

The load address, execution address, length and file attributes from the named file's catalogue entry are read into the parameter block.

On exit, A contains the file type:

0	indicates Not found
1	indicates File found
2	indicates Directory found (ADFS only)
&FF	indicates a protected file (ie E attribute set) (ADFS only)

A = 6 (&06) Delete the named file.

The information in the named file's catalogue entry is transferred to the parameter block and then deleted from the catalogue.

A = 7 (&07) Create an empty file.

The size of the empty file is determined by the start address and end address entries in the parameter block but no data is transferred. It is usually convenient to set the start address bytes to zero and use the end address bytes to define the length of the file.

A = 255 (&FF) Load the named file into memory at a location determined by the content of parameter block byte XY+6:

If XY+6 is zero, the file is loaded into memory at the address specified in XY+2 to XY+5.

If XY+6 is non-zero, the file is loaded into memory using the files own load address (see section J.10).

On exit : A is undefined (except for OSFILE 5)

X and Y are preserved

C, N, V and Z are undefined

The interrupt status is preserved but may be enabled during the call.

Note

Although four bytes (XY+14 to XY+17) are allocated for the attributes associated with an object, only the least significant four bits of XY+14 have any meaning under DFS or ADFS:

bit	Meaning when set
0	R attribute set (ADFS only)
1	W attribute set (ADFS only)
2	E attribute set (ADFS only)
3	L attribute set

OSWORD

Three OSWORD calls are recognised by the Disc Filing System and four by the Advanced Disc Filing System. In each case, the call number is supplied in A and X and Y must point to a control block in memory, details of which are given in each description.

Call address &FFF1
Indirected via &20C (WORDV)

Actions specified by A :

A = 112 (&70) Read Master Sequence number and status byte (ADFS).

XY points to a two-byte block in memory. On return, XY contains the Master Sequence number for the current directory (in binary-coded decimal (BCD) format). XY+1 contains a status byte; bits set have the following significance:

bit	Meaning when set
0	File ensuring in progress (IRQ pending)
1	Bad free space map
2	*OPT1 setting
3	undefined
4	undefined
5	Winchester disc controller present
6	Tube in use by ADFS
7	Tube present

A = 113 (&71) Read free space (ADFS)

XY points to a four-byte block in memory. On return, the 32-bit value of the available free space (equivalent to the value output by `*FREE`) is placed in this block.

A = 114 (&72) General read/write function (ADFS)

XY points to a 15-byte parameter block with the following format:

XY	zero
XY+1	<LSB of pointer to data in memory>
XY+2	.
XY+3	.
XY+4	<MSB of pointer to data in memory>
XY+5	<&08> to read; <&0A> to write
XY+6	bits 5–7 <drive> (see below) bits 0–4 <5 high order bits of absolute sector number>

XY+7 <8 middle order bits of absolute sector number>
XY+8 <8 low order bits of absolute sector number>
XY+9 <sector count for read operations>
XY+10 unused
XY+11 <LSB of data length for write operations>
XY+12 .
XY+13 .
XY+14 <MSB of data length for write operations>

Bits 5–7 of XY+6 are ORed with the current drive number to give the drive number to be accessed. The absolute sector number is a 21-bit value, high order bits first.

On exit, XY contains 0 if the operation was completed successfully; any other value indicates a disc error, typically:

Error code	**Meaning**
72 (&48)	Cyclic Redundancy check error
80 (&50)	Sector not found
96 (&60)	Bad command
97 (&61)	Bad address
99 (&63)	Volume error
101 (&65)	Bad drive

A = 115 (&73) Read last error information (ADFS)

If this call is made immediately after a disc error of some kind (including a data error in sequential filing), error information is returned in a 5-byte control block with the following format:

XY <8 low order bits of absolute sector number>
XY+1 <8 middle-order bits of absolute sector number>

XY+2 bits 5–7 <drive number>
bits 0–4 <5 high order bits of absolute sector number>
XY+3 Disc error number (see below)
XY+4 Channel number of file where error occurred

Only one of the contents of XY+3 and XY+4 will be valid for a given type of error. Where a disc error number is appropriate, the top bit is set if the absolute sector number for the operation was valid. The channel number (where appropriate) is given in hexadecimal.

A = 125 (&7D) Read Master Sequence number (DFS)

XY points to a single byte in memory. On exit, the specified byte contains the Master Sequence number (in binary-coded decimal (BCD) format) for the current drive.

A = 126 (&7E) Read disc size (DFS)

XY points to three-byte block in memory. On exit, the block contains the total number of bytes associated with the current drive (LSB first):

40-track : &19000
80-track : &32000

A = 127 (&7F) General read/write function (DFS)

XY points to a 10-byte parameter block with following format:

XY <drive number>
XY+1 <LSB of pointer to data in memory>
XY+2 .
XY+3 .
XY+4 <MSB of pointer to data in memory>

XY+5 3 (see below)
XY+6 <&53> to read; <&4B> to write
XY+7 <track number>
XY+8 <sector number>
XY+9 bits 5–7 <size of sector in bytes>
bits 0–4 <number of sectors> (see below)
XY+10 result (see below)

XY+3 contains the number of parameters associated with the 'command' specified in XY+6 – this value will always be 3 for general read/write operations.

The sector size in bits 5–7 of XY+9 is a coded value which denotes the number of bytes in each sector of the disc:

bit 7	**bit 6**	**bit 5**	
0	0	0	128 bytes/sector
0	0	1	256 bytes/sector
0	1	0	512 bytes/sector

etc.

All DFS-format disc sectors contain 256 bytes.

Bits 0–4 of XY+9 contain the number of sectors to be read/written by the call.

On exit from a read or write operation, XY+10 will contain zero if the operation was successful or a disc error number, typically:

Error code	**Meaning**
12 (&0C)	Cyclic redundancy error (ID)
14 (&0E)	Cyclic redundancy error (data)
20 (&14)	Track 0 not found
22 (&16)	Write fault
24 (&18)	Sector not found

Index

Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
England