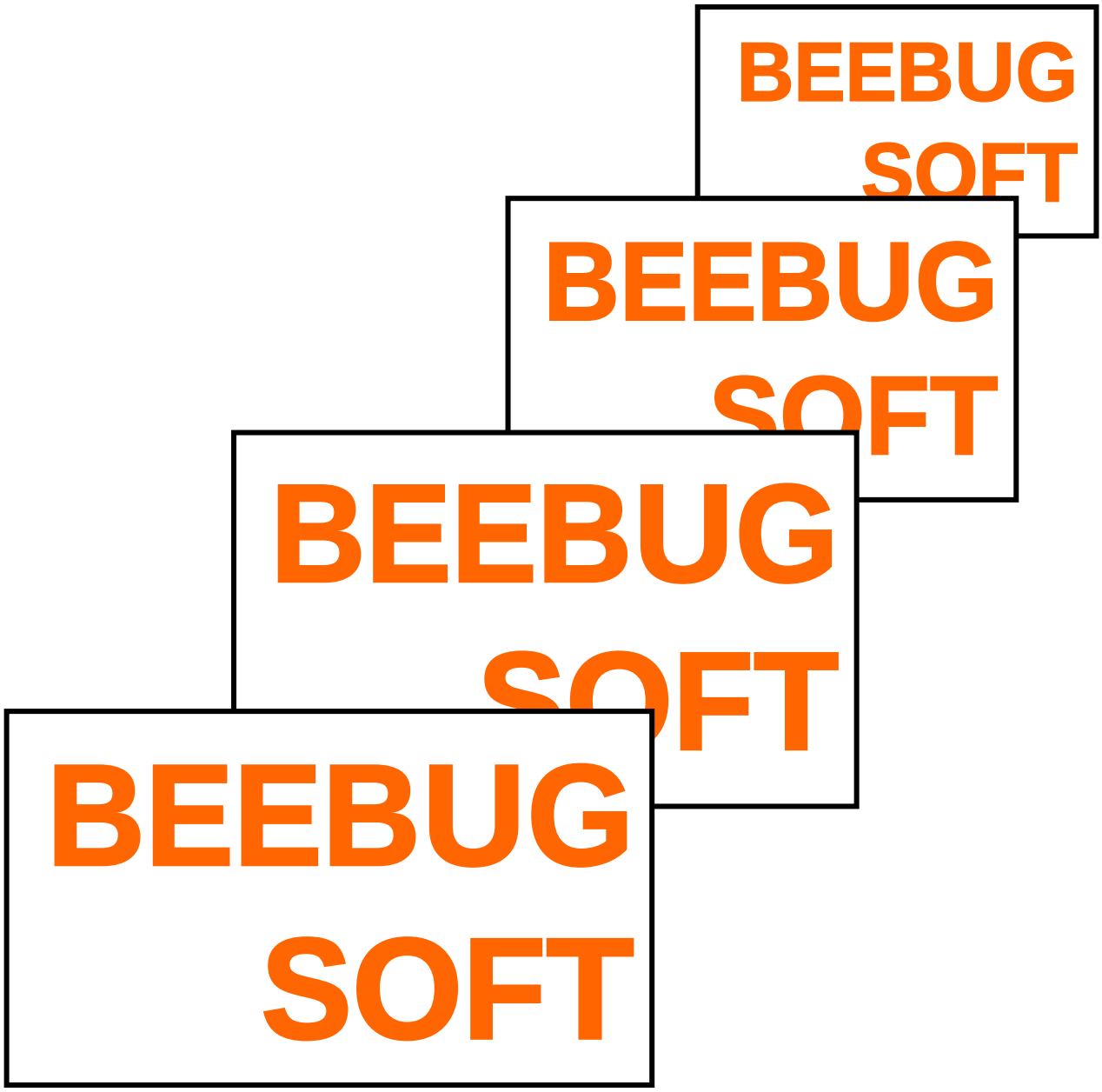
EXMON II

Extended machine code monitor supplied on Eprom for the BBC Micro and Electron



EXMON II

THIRD
GENERATION MACHINE
CODE MONITOR

For the
BBC MICRO
& ELECTRON

P.O. Box 50, St. Albans, Herts.

EXMON II

THIRD GENERATION MACHINE CODE MONITOR
For the BBC MICRO &ELECTRON

by
Mark Tilley

CONTENTS

1.	Introduction	3
2.	Starting Instructions	5
3.	General Overview	6
4.	Memory Editor	11
5.	Simple Commands	13
6.	Relocation	15
7.	Debugging Commands	17
8.	Single-Stepping	19
9.	Trace Options	20
10.	Dual Screen Operation	21
Appendix (i)	Examples	24
Appendix (ii)	Assembling from tape/disc	26
Appendix (iii)	Electron Notes	27
Appendix (iv)	Command Summary	28

1. INTRODUCTION

EXMON II is an extremely sophisticated third generation machine code monitor for the BBC Micro and Electron. It incorporates a number of advanced features not to be found on EXMON I (i.e. the first release of EXMON), while omitting only one EXMON I feature -- the on-screen Help facility. This had to be sacrificed because of lack of space, and is replaced by a command summary card. Electron users should consult appendix (iii) at this point.

EXMON II incorporates all the features that one would expect in a machine code monitor, such as display memory, disassemble, verify, fill memory, string and byte search, and a powerful program relocator. In addition it contains a new on-screen editor, a programmable front panel displaying in hex and Ascii, or disassembly, and a set of sophisticated debugging tools. These include single-stepping, break points, a trace facility, and dual screen operation (Dual screen operation is not implemented on the Electron version). A special facility allows parameters to be entered in hex, decimal, or as a general expression e.g. 44*(2 ^ 5-1) including the use of variables and label names used in your assembly listing.

The Editor

The Editor offers on-screen editing with full cursor control and bi-directional scrolling. The Editor displays in hex and Ascii format, and to alter a memory location, you simply overtype on screen. Pressing Tab puts the Editor into Assembly mode. The screen displays a full disassembly, and new assembler instructions may be entered from the control part of the screen. Tab will instantly revert to the hex and Ascii mode, while Copy will reset the disassembly to the address currently being edited.

Debugging and Dual Screens

EXMON II offers four major facilities to aid the debugging of machine code. Firstly it provides the usual breakpoint facility, allowing the user to run his code up to a predefined point, then examine and alter any of the 6502's registers, and any memory locations before proceeding. In addition, a conditional breakpoint or Trace facility is implemented. This allows a variety of program Run modes, with conditional breakpoints set on file value held in the 6502 accumulator or in the X or Y registers, and with continuous display of the code currently executing.

EXMON II also provides a simulation mode, allowing single-stepping through a program one op-code at a time. Before each line is executed, it is displayed on the screen, as is the state of the 6502's registers. Any register or memory location may be altered at any point in the simulation, providing very quick debugging of code.

EXMON II's dual-screen facility adds a further dimension to the single-step and trace options, though this has not been implemented on the Electron version because of the large amount of memory that it would require. When the dual-screen is enabled the user may opt to have all VDU output of his program displayed on the user screen, and at the press of a key, return to EXMON II's control screen for status readout, program listing, register alteration etc. EXMON II provides three dual-screen options during single stepping, allowing either the alternate display of the user's program screen and the control screen with each program step; or the continual display of either control or program screen. In any state, the two screens may be toggled at the press of a key.

This brief summary of the features offered by EXMON II will give some indication of its sophistication. There are more than 60 different commands associated with EXMON II, and you will need some practice before mastering them to the full. To help with the process of familiarisation, we have included a command summary card for quick reference use, and the back of this manual contains a fuller summary of all commands. These are grouped under a number of headings which reflect those used in the manual. The debugging and dual-screen sections are situated at the end of the manual, and it is advised that you familiarise yourself with sections 2-5 before attempting these.

It should be noted that EXMON II is intended primarily as a tool for machine code programmers, and thus for people with a certain familiarity with associated matters. If you are just embarking on the enjoyable experience of machine code programming, then EXMON II will prove invaluable both as a too!, and as a self teaching aid. The single stepping and dual-screen facilities will allow you to test pieces of code that you have written and to watch their immediate effect, both on screen, and on the registers and stack of the 6502, and in any other designated area of memory. It is recommended that beginners to machine code consult one of the many suitable works on the subject. One which we would particularly recommend is Assembly Language Programming for the BBC Microcomputer by Ian Birnbaum, published by Macmillan.

2. STARTING INSTRUCTIONS

(i) Fitting

EXMON II is supplied in an 8k Eprom, which should be fitted in one of the "paged Rom" sockets in the computer. It should be installed to the left of the Basic Rom. See separate fitting sheet for full installation instructions.

(ii) Calling EXMON II

Once the Eprom has been installed, it should announce itself at power-up, and can be entered with the command *EXMON <return>, where <return> means press the Return key. This can be abbreviated to *E <return> (note that there is no full stop after the E. An optional parameter may be added to define work space; this is dealt with in section 3 (ix).

(iii) Leaving EXMON II

To quit EXMON II, type Q <return>, or use CTRL-Break. Using Break from within EXMON will not exit EXMON, but will perform a warm start.

(iv) EXMON and Basic

Because EXMON II makes use of certain routines in the Basic Rom (compatible with Basic I and II), it is desirable that Basic should be present in the machine and also initialised. You can ensure this by always calling EXMON II from Basic (rather than some other language Rom such as Wordwise etc.) -- or if not, at least make sure that Basic has been entered since power up. If Basic is not present, or has not been initialised, EXMON's 'C', '#" and Assemble commands will not function. Otherwise it will behave as normal.

(v) EXMON Version Number

When in Basic, typing *HELP followed by Return will give the normal Help message: this should now include EXMON it with its version number. Please quote the version number in any correspondence.

3. GENERAL OVERVIEW

(i) The panel

On entering EXMON II the panel will be displayed. This consists of two parts; the top line which lists the 6502 registers, and about half a page of memory listing. The register values shown can be altered at any time, and are restored when a program is executed or simulated. The flags of the status register that are set are shown individually. Also the top 4 bytes of the stack (if any) are displayed, with the top byte on the left.

The remainder of the panel shows an area of memory, which can be set to start at any address. By default the memory at the 'current address' (see (iv) below) is displayed. The 'P' command can be used to change this. On first entering EXMON II the panel contains a hex memory dump. This can be changed to a disassembly by pressing Tab. Once this is done, the current panel format becomes 'disassembly' and remains so until switched back to the 'hex dump' by pressing Tab again. The implications of the panel format will be seen later. This screen window is sometimes used for other purposes; when this happens the panel can be restored by pressing Return. The size of the window can be set to between 2 and 17 lines with the 'WI' command, though with a window of 17, only three lines are left for command entry,

Note: the complete panel is updated after each command, so any changes caused by the command will be immediately visible.

(ii) Commands

EXMON II commands consist of one, two or three characters, followed in most cases by a number of parameters. The command summary at the end of the manual gives a detailed description of the precise syntax of all the commands. Throughout the rest of the manual the following simple means of describing parameters will be used:

<address> means a 2 byte hex address, e.g. 81C7.

<range> means two 2 byte hex addresses defining a range of memory, e g. 8000 80EF

<byte> means a single byte hex number, e.g. 3A.

The Delete key will only allow the current parameter to be deleted. To delete further back along a line, press Escape.

The Escape key will abandon the current command entry.

(iii) Numbers

The numbers used by EXMON commands can be entered in several ways:

(a) as a simple hex number. Do not terminate the number with Return unless the number has less than the maximum number of digits, e.g. if entering 8000 as an address do not press Return, but do so for an address such as 900. Do not prefix with &, hex is assumed, e.g. E 1900 will enter the editor at address 1900 hex.

(b) by default. Pressing Return alone causes an appropriate default value to be substituted. The effect of this depends on the command and is not allowed in some cases. In general, though, the first address parameter will default to the current address (see (iv) below); the second address will default to &FFFF; and a single byte parameter will default to zero.

(c) as an expression, using the '#' option (see (v) (b) below. This allows decimal values to be input.

(d) as a relative address. This applies only where a <range> parameter is required, and enables the second address to be given as a displacement from the first. This is entered by prefixing the second number with '+', e.g. 6000 +FF is equivalent to 6000 60FF. The displacement may be given as an expression as in (c)

(iv) The current address

This current address is used for default values (see (iii) above), and also the address to start execution or simulation of a program (see section 8). It is displayed at the top of the screen as 'PC' and altered by the '@ ' command.

(v) Expression evaluation

(a) The 'C' command causes evaluation of an arithmetic expression, displaying the result as a 4 byte hex integer. In addition the least significant 16 bits are displayed as a positive decimal integer (so, e.g. -10 is displayed as 65526). The expression may contain +, -, *, /, ^, brackets, and most useful of all, Basic variable names. In particular this includes labels used in the source code (provided they are still defined). Note: Due to a bug in Basic I and II, brackets can only be nested to 16 levels.

Note: the expression evaluator uses Basic interpreter routines, so the Basic convention of assuming decimal applies. So hex numbers must be prefixed with & in this case. If there is no Basic Rom installed, attempting to call the expression evaluator will result in an error message to this effect. EXMON II is compatible with versions 1 and 2 of BBC Basic in this respect.

Note: the expression evaluator will not work unless Basic has been initialised, so if EXMON has been called from some other Rom it may not work. In this situation first use the EXMON II command 'N' (N for new) which resets the Basic pointers.

(b) The '#' option. This allows fully general expressions to be entered for any parameter of any command. Simply prefix the expression with '#'.

Examples:

D #start #start+22 will disassemble 23 bytes from the memory location given by 'start' (which may be a source code label).

@ #16041 will set the current address to decimal 16041.

Even expressions such as 4*(25/6+1) etc may be used, but expressions should not contain Basic keywords.

(vi) Operating system commands

All operating commands (prefixed with *) are accepted by EXMON, so it is possible to *LOAD, *CAT, etc. all from within EXMON. In particular *KEY can be used to set up often used command sequences, e.g. *KEY 0 E3000 II M<return>, will set up key 0 to enter the editor at &3000.

(vii) Paged Roms

Although EXMON resides at the same address as the other paged Roms it is capable of listing and disassembling them. To do this, first select the required Rom by entering ! <rom-id> where <rom-id> is the number between 0 and F (hex) specifying the position of the Rom. Once a Rom has been selected in this way all commands that use memory between &8000 and &BFFF will apply to this Rom. The number of the currently selected Rom is permanently displayed in yellow at the top of the control screen as a digit prefixed by the ! character. After a cold start the Basic Rom will be selected regardless of where it resides.

Note: to discover the Rom id of the various Roms in your machine, enter P8000 to set EXMON's front panel to the area of sideways Rom carrying copyright messages, then select Rom ids in turn. i.e. enter !0 then !1 up to !F. When a Rom is present its name should appear in the first few lines of characters at the far right of the panel.

(viii) Printing

When in the control screen, EXMON intercepts all control codes so <CTRL-B> will not switch the printer on. Instead a command is included to give the same effect.

Typing 'H' (H for hardcopy) toggles the printer on/off. When 'on', the output of all commands will be sent to the printer, and as an indication, the title line of the control screen will flash.

(ix) Memory usage

EXMON II uses just one page (i.e. 256 bytes) of Ram as workspace. The page used is indicated in yellow at the far top left of the control screen, and this can be altered at any time if it clashes with some other use. By default page 05 is used, but any page between 4 and &7B (except 8 and &D) can be used instead. For example to use page 09 call, EXMON with *EXMON 9 (or *E 9) to change to page 09 from within EXMON, enter 'WS 9'. In addition EXMON II uses 2 other bytes, &28A and &28B. All other memory (including all zero page locations) can be used freely and will not be affected by EXMON.

The user should note that choosing ill-advised workspace areas (e.g. anywhere in Rom or in screen memory -- &7C00 -- 7FFF) will have unpredictable effects.

(x) Warm and cold starts

After EXMON has been called once it may be 'warm started', thus preserving all saved register values, panel display, Rom selection, etc. Entering with '*EXMON' will cause a warm start. Pressing BREAK when in EXMON will cause a warm start, even retaining values of variables. To force a cold start either call EXMON specifying the workspace, e.g. '*EXMON 5', or use the command 'N' <return> from within EXMON. The 'N' command also initialises Basic. Note that once the workspace page has been selected this will be remembered and used for all future calls of EXMON. This is not even reset by <CTRL-Break>.

(xi) Leaving EXMON

The command 'Q' will quit EXMON and return to Basic, leaving any Basic program and all variables intact. There is no need to type 'OLD' to recover the program. Typing *BASIC will cold start Basic, but will leave the EXMON window defined.

COMMANDS RELATING TO SECTION 3

*EXMON or *E	Call EXMON. This makes a cold start, unless EXMON has been previously called, in which case a warm start is executed.
*EXMON<byte> or *E <byte>	This forces a cold start and a re-allocation of workspace.

Commands from within EXMON

Break	Warm start EXMON
N<cr>	cold start of EXMON from within EXMON itself ('N'=NEW) A warning beep is given after 'N' is entered
Q	Quit EXMON, and enter Basic. EXMON may be re entered with *E giving a warm start
WS<byte>	Change EXMON workspace
H	Toggle printer ON/OFF (H="Hardcopy") The title line of the control screen will flash when in print mode. Note also that if no printer is connected and turned on, the screen will lock up. Either connect a printer or press Escape, then toggle H to turn off print output.
p=PANEL	Syntax: P <address> Effect: Sets the start address for the default panel listing. If no address is specified the panel will show memory starting from the 'current address'. Pressing Return when in command mode will restore the default panel listing. <Tab> will switch between display formats.
WI=WINDOW	Syntax: WI<byte> Effect: Sets the height of the panel window. <byte> must be between 2 and 17 lines dec or (11 hex). If a number is not specified the default window height (14 dec) is restored.

4. THE ON-SCREEN EDITOR (hex, Ascii and Assembler)

E = EDIT

Syntax: E <address>
Effect: Enter the on screen memory editor at the specified address, using the current panel format, <address> may be omitted, in which case the start address of the panel listing will be used.

This command incorporates 3 methods of altering the numbers stored in memory. These are

(i) by typing hex numbers;
(ii) by typing Ascii characters; and
(iii) by typing 6502 mnemonics which are then assembled.

The editor is entered using the current panel format. This means that if the panel contains a disassembly the line assembler will be entered, if a hex dump the hex/Ascii editor will be entered.

(i) The hex/Ascii editor

When the hex/Ascii editor is invoked (e.g. from a command such as E1900) the window will show a hex and Ascii memory listing, and the cursor will be initially positioned at the top left byte in the panel window. The values shown can now simply be over-typed. The cursor keys will move the cursor in any direction. Use together with Shift to display a page at a time.

The Ascii characters are displayed at the right of the screen. To edit these directly press Copy. The cursor will now be positioned in the Ascii part of the listing, and these can be altered by typing any characters (including control Codes). Copy will switch back to hex editing.

Tab will switch to the line assembler.

Escape will return to command mode.

(ii) The line assembler

When the line assembler is invoked the panel window will show a disassembly starting at the specified address. The command window will show that address with the cursor positioned after it, ready to assemble 6502 instructions that are input. Only one instruction per line will be read. Labels can be used in the normal way. As the code is assembled the panel will be immediately updated, but will not scroll, so the previous instructions are not lost. If the assembled code is longer than one windowful, press Copy to restart the display at the current assembling address.

The up and down cursor keys can be used to move forward or backwards, and together with Shift to move in jumps as above. Return will move on one byte, while <cursor-down> will move one instruction.

Tab will switch to the hex editor.
Escape wiII return to EXMON command mode.

Note that it is not possible to overwrite Rom using the editor, so that if you try to edit a Rom address, e.g. &8000 onwards, the display will revert back to its previous value each time an attempted 'write' is performed.

5. OTHER MEMORY, EDITING AND MANIPULATION COMMANDS

D = DISASSEMBLE

Syntax: D <range>

Effect: Disassembles the given range of memory. The output appears in the panel window, one page at a time. Press space bar for each new page. The listing shows standard 6502 mnemonics, hex and Ascii. '???' signifies an unrecognised opcode.

L = LIST

Syntax: L <range>

Effect: Lists the given range of memory in hex and Ascii, in the same way as the D command. With both of these commands, if the start address is omitted (by pressing Return alone) the current address is used. If the end address is omitted &FFF is used.

K = DISASSEMBLE & SAVE

Syntax: K <range> <filename>

Effect: Disassembles the range of memory, spooling to the specified file. The first record of the file will contain the Basic command 'AUTO', so that when the file is *EXEC'd from Basic, a Basic program will be created. This can be edited and run to reassemble the original section of code.

Notes:

(i) Branch instructions are disassembled with relative, rather than absolute, addressing.

(ii) The filename is restricted to 7 characters, for compatibility with the DFS.

S = SEARCH

Syntax: SS <range> <string>
or SB <range> <bytes>

Effect: Searches the specified range of memory for the Ascii or hex string. At each find, the panel window will display the relevant area of memory. Pressing Escape will abort the search leaving the panel display; pressing other keys will continue the search.

The search input may be up to 25 characters long for an Ascii search, or 25 bytes for a hex search. In either case it may contain up to 5 wildcards (characters that will be matched by any byte. To signify a wildcard type '@'. If too many wildcards are entered, an error message will be given. Terminate the input by pressing Return. Do not enter quotation marks at the end of a string.

F = FILL

Syntax: FS <range> <string>
FB <range> <bytes>

Effect: Fills the specified range with the Ascii or hex string. No default is allowed on the end address. On completion the panel window will show the start of the area of memory filled.

M = Move

Syntax: M <range><address>

Effect: Copies a block of memory to the memory starting at <address>. This is an 'intelligent copy' in that it copies with overlapping ranges.

V = Verify

Syntax: V <range>< address>

Effect: Compares the memory block <range> with the block starting at <address> and lists any differences.

O = OSBYTE or OSWORD call

Syntax: OB <byte><byte><byte>
or OW <byte><byte><byte>

Effect: Calls OSBYTE or OSWORD, putting the 3 bytes in A, X, Y respectively. The values of A, X and Y on exit are displayed. The saved registers are unaffected.

6. RELOCATION

The relocator will take a machine code program in memory, move it to another specified location and attempt to correctly adjust the program so that it will run at the new location. There are several reasons which may make it impossible to relocate a given program, so programs that will need to be relocated must avoid these:

(i) the interspersion of data throughout the program (like the way the BBC O.S. handles error messages). Data should be contained in a few continuous blocks.

(ii) references to addresses within the program as immediate operands, or in look-up tables. For example if a program uses an action address table, which gives addresses within the program. The action addresses could be used as follows:

```
LDA AAL,X      get action address low byte
STA M1
LDA AAH,X      get high byte
STA M2
JMP (M1)
```

This sequence is quite unrelocatable because the addresses are stored simply as data. Similarly, using LDA # address(hi-byte): STA M1: LDA # address(lo-byte): STA M2, where 'address' is a source code label, is unrelocatable.

The way to avoid this problem is to store all such addresses in a relocatable address table. Store them as operands of instruction within the program, e.g., LDA addr1: LDA addr2: LDA addr3. The LDA instruction is merely a dummy which ensures that the addresses are relocated.

R = RELOCATE

(i) Syntax: R <range> <address>

Effect: Relocates a program given by <range> to the new address. This form assumes the <range> contains pure program, with no data. The exact effect is to adjust all addresses in the program that lie within the range of the program. This form is only of use if the relocated program is to use the same workspace.

(ii) Syntax: R <range> <address> <up to 3 subranges>

Effect: The subranges (lying within <range>) specify areas of memory used for data, and which are therefore simply to be moved, not relocated. The rest is relocated as above, but now any references to addresses within the moved data blocks will also be relocated. If illegal subranges are entered, or they are entered out of order, 'input error' will be given.

RELOCATOR ERROR MESSAGES

Relocation will still occur even if errors are encountered, because most practical relocations will involve errors that can be corrected manually after relocation. There are 4 error messages that can be produced by the relocator.

Data at hhhh — Given if an unrecognised opcode is encountered in a relocation section. This would usually mean there is data mixed with program, or that the. subranges were incorrectly specified.

Split at hhhh — Given if the start of a 'move-only' subrange occurs in the middle of an instruction.

Outside range at hhhh

Relative jumps need no relocation, so if one jumps outside the range of relocation it will be meaningless in the new version.

Overflow at hhhh — This unusual error occurs if a program residing in zero-page is relocated elsewhere, so some one byte zero-page operands need two bytes when relocated.

7. DEBUGGING COMMANDS

(i) Commands to execute a program

G = GO

Syntax: G <address>

Effect: The saved values are restored to the 6502 registers and execution begins at <address>, which defaults to the current address. Breakpoints are inserted before execution.

In addition to saving the registers, the following variables are saved by EXMON and restored whenever a program is executed:

Input stream (set by *FX 2)
Output stream (set by *FX 3)
Cursor editing status (set by *FX 4)
Auto-repeat delay (set by * FX 11)
Escape key status (set by *FX 200).

The current ROM (selected by the '!' command) is paged in before execution.

J = JSR

Syntax: J <address>

Effect: As above, but returns to EXMON on 'RTS' instruction. This is useful for testing subroutines in isolation. The registers are saved and displayed on return. Breakpoints are not inserted.

(ii) Commands to set register values

A	<byte>	set accumulator.
X	<byte>	set X register.
Y	<byte>	set Y register.
IS	<byte>	set stack pointer. This may not be set to below &6F. The default value is &FF (i.e. empty stack). EXMON will then preserve all memory locations from &101+SP to &1FF, and these may be edited directly using the 'E' command.
IP	<byte>	set processor status (PSW). The flags may also be individually inverted with the 'I' command, e.g. 'IC' inverts the carry flag.
@	<address>	set current address / program counter.

(iii) Breakpoint commands

Breakpoints are used in conjunction with the 'G' command to stop the program at various points in order to keep track of what the program is doing. When a breakpoint is reached, EXMON will save the register contents (before executing the instruction at the breakpoint address), and return to the monitor. The register and memory contents can then be examined and altered as required to test the program, The 'G' command can then be used again to continue to the next breakpoint.

Up to 10 breakpoints may be set at any one time, and their addresses are displayed at the bottom of the screen

BS = SET BREAKPOINT

Syntax: BS <address>

Effect: Puts breakpoint at specified address (default is current address). This MUST be the address of the first byte of an instruction. An error message will be given if the byte in question is not a valid 6502 epode. Note that with an instruction such as JSR &2020 [20, 20, 20], no error message would be given if a breakpoint is put at the middle byte, because it happens to be a valid opcode. When executed a breakpoint will not occur, instead the instruction will be altered to [20, 0, 20], which is JSR &2000.

BC = CLEAR BREAKPOINT

Syntax: BC <address>

Effect: Removes breakpoint. This may be used with default address to delete the current breakpoint before resuming execution.

BW = WIPE BREAKPOINT TABLE

Syntax: BW

Effect: Remove all breakpoints.

Further notes on use of breakpoints:

(a) The effect of the 'BS' command is to store the address in the breakpoint table; the program is not altered so it will be disassembled correctly. The breakpoint is only put into the program (a zero byte replacing the byte at the breakpoint address) when the 'G' command is issued.

(b) Permanent breakpoints (i.e. zero bytes in the program) have the same effect as EXMON breakpoints.

(c) VERY IMPORTANT - Breakpoints must only be placed at the first byte of an instruction.

(d) All breakpoints are cleared when Break is pressed.

8. SIMULATION - SINGLE STEPPING

As an alternative to using breakpoints EXMON will simulate the execution of a program, one instruction at a time. This enables a small part of a program to be examined in detail. It also allows the debugging of programs located in ROM.

<spacebar> = SINGLE STEP

Syntax: <spacebar>

Effect:

(a) If in command mode, enter simulation mode at current address. The next instruction and the register contents will be displayed but the instruction not executed.

(b) Subsequently, executes current instruction, displays the register contents after execution and displays the next instruction. A running display will scroll in the command window; the top line will always show the correct register and stack contents.

Any operating system call, such as JSR &FFEE, will be executed as if one instruction. EXMON is not, in general, capable of simulating operating system routines. These should be executed as a single instruction using the "/"' below.

/ = SINGLE STEP ON ONE LEVEL

Syntax: /

Effect: As for single step command above unless the current instruction is a 'JSR' call, when it is executed as one instruction. This is useful when certain subroutines are known to be working. This only works after simulation mode has been entered.

Note: while in simulation mode any other command can be entered normally, causing automatic return to command mode. In particular the register commands can be used immediately to simulate particular conditions. Pressing the space bar after this will reinstate simulation.

<DELETE> = SKIP

Syntax: <DELETE>

Effect: The current instruction is skipped, and not simulated. This only applies after simulation mode has been entered.

9. SIMULATION – THE TRACE OPTIONS

Simulation can also be used in conjunction with breakpoints and conditional breakpoints to provide a trace facility. In this case, the same breakpoints as used by the 'G' command are employed, but the program code is not altered. Instead the program counter is compared with the addresses in the breakpoint table after each instruction is simulated. This allows breakpoints to be 'put' into ROM,

TB = TRACE FROM GIVEN ADDRESS UNTIL BREAKPOINT

Syntax: TB <address>

Effect: Simulates instructions continuously (as if the space-bar were pressed many times), displaying each. The address entered after TB is the start address of the trace operation. If it is omitted, the Trace will start at the value of PC. Pressing Escape will immediately exit from this command. When a breakpoint is reached, a beep is sounded and simulation mode is entered, ready to continue by single stepping. Alternatively, of course, another 'TB' command may be issued.

TSB = TRACE FROM GIVEN ADDRESS UNTIL BREAKPOINT, SUPPRESSING LISTING
Syntax: TSB <address>

Effect: As 'TB' except that the instructions and registers are not listed. When a breakpoint is reached, EXMON reverts to command mode.

TA, TX, TY = TRACE FROM GIVEN ADDRESS UNTIL REGISTER VALUE OCCURS

Syntax: TA <byte> <address>
or TX <byte> <address>
or TY <byte> <address>

Effect: As 'TB', stopping at breakpoints, but additionally stopping (and entering simulation mode) when the specified register becomes equal to the: value <byte>.

TSA, TSX, TSY = Trace until register value, suppressing listing

As TA, TX & TY, but with trace output suppressed.

10. DUAL SCREEN OPERATION*

When debugging programs with any visual output, most monitors -- including EXMON I -- are of only limited help. But EXMON II incorporates a dual screen facility which allows all program output to go to the screen, whilst retaining full control through a second screen. The two screens -- EXMON's normal control screen and the user's program screen -- can then be toggled as desired.

The dual screen facility can be used in combination with a number of program-running options:

(i) 'G' with optional breakpoints
(ii) Single stepping and simulation
(iii) The various trace options

Dual screen operation is achieved in EXMON II by saving part of the user's program screen in a predesignated 1.25k of workspace when in dual screen mode. The user may set this workspace to any page boundary within RAM, including sideways RAM. The area of RAM designated for this purpose is displayed at the top RHS of EXMON's front panel. The default value is 12 (i.e. start address 1200 hex). The user should take care not to set this to memory areas designated for other purposes.

To initialise the dual screen format, the command ZI is issued. ZI may be followed by a new workspace parameter. E.g. ZI 77 would initialise dual screen operation with a workspace of 7700 hex. Once the command ZI has been issued, a 'CP' will also appear at the top right of the screen.

These two letters indicate the dual screen status for simulation. With both C and P displayed, EXMON II will flip between the Control screen and the user's Program screen during simulation (i.e. with each press of the space bar or '/' key). In fact two presses of these keys are now required during simulation -- one to perform the command and display the updated program screen, the other to return to the Control screen.

The dual screen status may be altered from the Control screen with CTRL-Z. This toggles between three possible states CP, C and P. Toggling from CP (the default state) causes a C to appear in the top right hand corner of the screen, indicating that during simulation, EXMON will remain in the Control screen. It is possible to view the program screen at any time by using CTRL-Tab, or by reverting to CP status and continuing to single step. In Control-screen only mode however, the user's program will not update the program screen. This is not physically possible.

A further press of CTRL-Z moves the status indicator to P only. Here the program screen remains displayed during single stepping, and is updated with each step. The Control screen is reinstated at the end of a program run sequence -- e.g. at the end of a Trace, or when a breakpoint is reached etc.

Footnote:

* Not available on the Electron version of EXMON II. See Appendix (iii).

Program Screen.

At no time when the program screen is displayed is it possible to issue new control commands. This must be accomplished by first switching to the Control screen by pressing the Escape key.

When the Program Screen has been entered from the control screen by pressing CTRL-Tab, the user may issue any normal keyboard commands (note that this is not possible when the program screen has been displayed by using the space-bar in simulation). Keys will print to the screen directly, and CTRL commands may be issued.

E.g. CTRL-H move cursor back one space
CTRL-I move cursor forward one space
CTRL-J move cursor down one line
CTRL-K move cursor up one line
CTRL-L clear screen
CTRL-B printer on
and so on.

You may also change screen modes from the program screen using CTRL-V followed by the mode number required. The default mode is 7. Remember however that, as usual, when you change mode with CTRL-V, HIMEM is not reset, and you may easily overwrite any Basic program or assembler source code or variables stored in the machine, and conversely the presence of program or variables in the area of memory normally used by the screen, can cause dot patterning to appear on the screen.

HIMEM can be reset either by:

(i) Quitting EXMON II (Q), typing the required MODE (e.g. MODE 1 <return>) and re-entering EXMON II with * E

(ii) Poking the known value of HIMEM directly into page-zero at &6 (low byte), &7 (high byte).

The commands are as follows:

ZI = INITIALISE DUALSCREEN

Syntax: ZI <byte>

Effect: Sets up the second screen (initially a blank mode 7 screen) which will subsequently be restored when a program is executed. The <byte>, which is optional, specifies the first page of memory to be used for storage. Five consecutive pages are required. The default storage area is at &1200, which is ideal for DFS users (unless several files are being used). If sideways RAM is installed, this may be used for the dual screen storage area. If <byte> is &80 or higher, EXMON prompts for the position of the sideways RAM.

ZC = CANCEL DUAL SCREEN

Syntax: ZC

Effect: Cancels dual screen operation.

<CTRL-Tab> = DISPLAY PROGRAM SCREEN

Syntax: <CTRL-Tab>

Effect: The saved screen is restored. While displayed any characters typed (including control codes) affect the screen in the normal way. Thus the screen can be edited by moving around with CTRL-H, CTRL-I, CTRL-J, and CTRL-K.

Escape will return to the control screen; the program screen will be saved with any changes.

CTRL-Z Toggle between the three modes of dual screen operation, as indicated at top RHS of screen.

CP indicates full dual screen operation with simulation showing both screens at each step. A double press of the space bar is needed for each step.

C indicates control screen only -- program screen will not be updated during simulation.

P indicates program screen only -- program screen continuously updated, but no display of control screen.

Output when dual-screen is not used:

If a program is executing (as opposed to being simulated by EXMON) any output will occur as normal.

When in simulation mode, output is intercepted by EXMON to remove control codes, and the remaining output is displayed in the panel window. The exceptions to this are the 'TS' commands (i.e. 'TSB', 'TSA', 'TSX' or 'TSY') In this case output is sent to the command window, thus allowing the panel to be used (for example to keep a memory listing on display).

When output is intercepted all control codes are ignored except 7, 10, and 13 (beep, line feed, carriage return).

APPENDIX I
EXAMPLE OF SINGLE STEPPING WITH DUAL-SCREENS AND A TRACE*

(i) Dual-Screen Single-Stepping

As an example of single stepping with dual screens, you may care to try the following:

*E	(Enters EXMON)
Zl<return>	Initialises dual screens (notice the CP appear top right)
E 3000	Enter Editor at location &3000
Tab	Selects assembler mode

LDA # 99 STA &7E00 LDA #100 STA &7E04 LDA # 101 STA &7E06	These 6 lines of code will place the letters c d and e on a mode 7 screen 99 is the Ascii value of c etc and &7E00 is a location on the screen. But nothing will happen until the code is actually executed during single stepping below.
@ 3000	Sets EXMON's program counter to the start address of the code

space bar	Initialises single step
space bar	User screen blank
space bar	A register loaded with &63 (i.e. 99dec)
space bar	User screen blank
space bar	&63 sent to screen
space bar	Letter 'C' appears on user screen
etc	etc

Footnote:

*Not applicable to the Electron version.

(ii) Example of a Trace

To illustrate the use of the trace command -- more precisely the TA command -– enter the code used in the previous example viz

```
3000    LDA #&63
3002    STA &7E00
3005    LDA #&64
3007    STA &7E04
300A    LDA #&65
300C    STA &7E06
```

We will Trace until the accumulator contains the value 101 or &64. To make sure that it does not contain this value before you start, enter

A 0 <return>

You may need to enter ZI 12 to clear the user screen if it has been used previously and left uncleared.

Now enter

TA 64 3000

or

TA #101 <return> 3000

You need the Return when entering the byte value in decimal, since EXMON does not know how many digits will be required -- in hex it is a maximum of two.

When you have entered this, a beep will he heard, to indicate that the Trace has been terminated because the stated condition was reached, and single-stepping from the control screen will be initiated. Further presses of the space-bar will step through the rest of the program. You will notice that the program screen was updated during the trace.

APPENDIX lI
USING EXMON II's ASSEMBLER TO ASSEMBLE FROM TAPE/DISC.

It is possible using EXMON's assembler to assemble code from tape or disc prepared on a word processor such as Wordwise. This provides all the convenience of a wordprocessor to edit and organise your assembly code, and should allow you to assemble code much greater in length than BBC Basic's resident assembler will allow.

The best way to illustrate the principle is with an example. Enter Wordwise editing mode, and type the following

E#&3000<ret> (no spaces here)
LDA #100<ret>
STA &7E00<ret>
LDA #102<ret>
STA &7E02<ret>

Return to the Wordwise menu, select option 8 to spool out the file, and save it under the name EXM say. Now exit Wordwise (e.g. CTRL-Break) and enter EXMON (with *E).

Set the panel to &3000 (P 3000), and get into assembly/disassembly mode by pressing Tab, then in return to the prompt, simply type

*EXEC EXM

Your listing will be EXECed in. The first command will call the editor at address &3000, and the assembler will be entered; as a glance at the panel (set this to &3000) will tell you.

EXMON's assembler will even allow you to use labels in the same way as the BBC assembler, and these may be used from Wordwise in the same way. This is also true of the operators EQUB, EQUW, EQUD and EQUS, providing that your machine is fitted with Basic II. These four pseudo operators are only implemented in Basic II, and EXMON uses routines resident in Basic for its own assembly options.

Note that if you wish to insert blank lines in your Wordwise assembly listing, these should contain a colon followed by Return. An unaccompanied Return character will cause the assembler to get out of key.

Note also, that because EXMON's assembler allows direct line input, it works in a single-pass mode, and will not allow forward labelling. The way around this is to EXEC in the Wordwise file three successive times. During the first pass EXMON will print 'no such variable' messages when it comes across forward labels: you should ignore these. On the second pass, no error messages should occur, but a third pass is necessary before all addressing is correctly handled.

APPENDIX III
ELECTRON NOTES

Users of the Electron version of EXMON II should note the following:

(a) EXMON II must be fitted external to the Electron in a sideways Rom socket. For further details see special fitting instructions.

(b) EXMON II for the Electron does not have the dual screen facility described in this manual. This is not viable on the Electron because of the large memory used for screen display in the most economical mode (6). Please disregard section 10, and the associated dual-screen commands given in the command summary.

(c) Please ignore all references to colour. In order to retain as much memory for the user as possible, mode 6 has been used for EXMON II's control screen. All display is therefore in monochrome.

(d) Please disregard the examples given in Appendix I, since these relate to dual-screens, and the display of characters in mode 7.

(e) Notwithstanding these necessary limitations, you will still find EXMON II a powerful aid in all machine code applications on the Electron.

APPENDIX IV
EXMON II COMMAND SUMMARY

1. General

*EXMON<byte>	Cold start EXMON, specifying workspace page.
*E<byte>	Cold start EXMON, specifying workspace page.
*EXMON	Warm start EXMON, with default workspace.
*E	Warm start EXMON, with default workspace.
Break	Warm start EXMON.
Escape	Abort command entry or execution.
N<cr>	Reset Basic pointers (NEW).
Q	Quit EXMON and enter Basic.
WS<byte>	Change EXMON workspace.
H	Toggle printer on/off.
*	Any operating system call.
C<expression>	Calculate value of expression.
OB<3 bytes>	JSR OSBYTE.
OW<3 bytes>	JSR OSWORD.
WI<byte>	Set window height.
P<addr>	Set default panel.
P<cr>	Set panel to default to current PC.
!<byte>	Change currently selected ROM.
<Tab>	Switch format (disassembly/hex dump).
<cr>	Return to default panel.
#	Decimal values and expressions follow.

2. Editor (hex, Ascii and Assembly)

E<addr>	Edit memory in current format.
<Escape>	Exit editor.
Hex. Editor:	
<Tab>	Switch to line assembler.
<Copy>	Switch between hex and Ascii editing.
<cursor-up>	Move up one line.
<cursor-down>	Move down one line.
<cursor-left>	Move back one byte.
<cursor-right>	Move forward one byte.
<Shift-cursor-down>	Move down one screenful.
<Shift-cursor-up>	Move up one screenful.
<Shift-cursor-left>	Move to top left.
<Shift-cursor-right>	Move to bottom right.
Line assembler:	
<Tab>	Switch to hex Editor.
<Copy>	Restart display at current address.
<cursor-up>	Move back one byte.
<cursor-down>	Move forward one instruction.
<Return>	Move forward one byte.
<Shift-cursor-up>	Move back 16 instructions.
<Shift-cursor-down>	Move forward one screenful.

3. Other memory Editing and Manipulation commands.

D <range>	Disassemble memory.
L <range>	List memory.
K <range><filename>	Disassemble and save to file.
SS <range> <string>	Search for Ascii string in given range.
SB < range> <bytes>	Search for string of hex. bytes.
FS <rangeA> <string>	Fill memory with Ascii string.
FB <rangeA> <bytes>	Fill memory with hex. bytes.
M <rangeA> <addrA>	Move memory block.
V <rangeA> <addrA>	Verify that two memory blocks are the same.
R <rangeA> <addrA> <0 to 3 rangeAs>	Relocate memory block, not relocating specified sub-ranges, otherwise adjusting all addresses in the range.

4. Debugging & simulation

@ <addr>	Set program counter.
A <byte>	Set accumulator.
X <byte>	Set X register.
Y <byte>	Set Y register.
IP <byte>	Set Processor Status register.
IS <byte>	Set stack pointer.
IC	Invert carry flag
IZ	Invert zero flag.
II	Invert interrupt flag.
ID	Invert decimal flag.
IB	Invert break flag.
IV	Invert overflow flag.
IN	Invert sign flag.
G <addr>	Execute program (GO).
J <addr>	JSR to <addr>.
<space bar>	Enter simulation mode. If already simulating then step one instruction.
/	When simulating, step one instruction, treating a JSR as a single instruction.
<Delete>	When simulating, skip next instruction.
TB <addr>	Trace from <addr> until next breakpoint.
TSB <addr>	Same, suppressing trace output.
TA <byte> <addr>	Trace until A= <byte> or breakpoint.
TSA <byte> <addr>	Same, suppressirig trace output.
TX <byte> <addr>	Trace until X=<byte> or breakpoint.
TSX <byte> <addr>	Same, suppressing trace output.
TY <byte> <addr>	Trace until Y= <byte> or breakpoint.
TSY <byte> <addr>	Same, suppressing trace output.
BS <addr>	Set breakpoint.
BC <addr>	Clear breakpoint.
BW	Wipe all breakpoints.

5. Dual Screen (BBC only)

ZI <byte>	Initialise dualscreen.
ZC	Cancel dualscreen operation.
<CTRL-Z>	Change simulationn display mode between CP; C and P
<CTRL-Tab>	Display program screen. Escape to return to Control screen.

6. Syntax definitions

<addrA> means an address with default by pressing <cr> not allowed.
<rangeA> means a range with default not allowed on the second address.

<addr>::=<addrA> <cr>(<cr> defaults to current PC address)
<addrA> ::=<4 hex. digits> <1-3 hex. digits><cr> #<expression>
<byte>::=<2 hex. digits> <hex digit><cr> <cr> #<expression>
(<cr> defaults to zero, except &FF for 'S' command)
<range> ::=<rangeA><addr><cr> (<cr> defaults to 'continuous listing')
<rangeA>::=<addr><addrA> <addr>+<addr>
<cr>::=<RETURN>
<expression>:: =general expression, involving +, -, *, /, ^, brackets,
variable names (e.g. assembly source code labels).