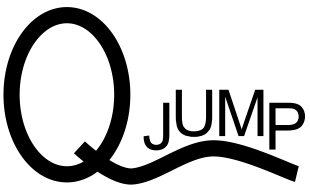
QMON II

Software by



Contents

0 An Overview

QMON II is a low level monitor/debugger designed specifically for the QL and its operating system QDOS. It is ideally suited to the task of checking and debugging assembly language programs, and extensions to the SuperBASIC interpreter. Even if you already have a monitor/debugger for your QL, QMON II will provide extra assembly language programming power. QMON can also be used to monitor programs written in high level languages.

QMON is designed to integrate into the QDOS environment. This makes it possible to monitor just one job in the QL, or all the jobs that are executing. While tracing a job, QMON will normally skip the entries into QDOS, but it can trace right through QDOS. QMON does not interfere with QDOS so it is entirely feasible to examine one job in the QL, while other jobs continue unimpeded.

QMON has an extensive range of facilities including a window based memory editor, single- and multi- stepping, tracing and back-tracing with fixed and conditional breakpoints, as well as a macro command facility for tracing and patching, together with an unusually powerful set of commands for examining and altering machine code and data.

Despite its wide range of facilities QMON is very compact. The program includes a complete MC68008 single line assembler, editor and disassembler, occupies only 11k bytes, while a reduced version is included which takes 5k bytes.

1 About QMON

QMON is a tool to assist software developers. It is not intended as an aid to pirating other people's software or circumventing any of the mechanisms for protecting software. For this reason, there are some facilities, which could have been included, which have been omitted. This should not affect the use of QMON for legitimate purposes.

QMON is supplied on a Microdrive cartridge or floppy disk with a number of files:

BOOT	SuperBASIC program to load QMON_BIN
QMON	SuperBASIC program to load QMON_BIN
QMON_BIN	the QMON resident debugger
JOB	SuperBASIC program to load JOB_BIN
JOB_BIN	job control extensions to SuperBASIC
CLOCKS	configurable clock program
CLOCKS_LIST	assembly listing of clocks

The job control extensions and the configurable clock are supplied primarily as examples for learning to use QMON. But the job control extensions are also valuable for program development. Before going any further please make a backup copy of the disk or cartridge, using WCOPY or similar (or our Transfer Utility!). Archive the original and use only the backup. This program is protected by international copyright law - do not break it.If we have anything to add to the manual, we will put a Quill UPDATES_DOC on the medium.

2 QMON Commands

Commands may be in either upper or lower case. In general a command consists of a one or two letter abbreviation, followed by an optional channel number (which specifies where the output, if any, from the command will be sent) and some parameters separated by spaces:

```
                    D1 28000 10
  Display   on channel 1   from address 28000   10 (hex) bytes
```

In most cases, most or all of the parameters are optional. Thus, after the example above, new defaults for the display command are set so that:

```
                    D1              has the same effect as
                    D1 28010 10
  Display   on channel 1   from next address   the same number
```

Note that, in QMON commands, a number is assumed to be hexadecimal to make the handling of data structures simpler, while the assembler assumes that a number is decimal. This is Motorola standard and it avoids confusion between, for example, register D2 and the address $D2. Throughout, QMON will accept hexadecimal numbers starting with $ (e.g. $28000, the system variable base) and decimal numbers starting with & (e.g. &131072, the base address of the screen). The command handling of QMON will accept simple arithmetic expressions in both hexadecimal and decimal, and there is a command to calculate the value of an expression and print it in both (unsigned) decimal and (unsigned) hexadecimal.

In this document, the parts of commands which are printed in upper case are the actual characters typed, those parts in lower case are symbolic while square brackets ([...]) are used to denote optional parameters. A lower case 'c' is used to denote the optional channel number.

<ESC> denotes the key marked ESC, <ENTER> denotes the key marked ENTER and <CTRL> denotes the key marked CTRL.

<ENTER> will cause a line to be actioned, while <ESC> and the up and down arrows will abandon the current line.

When typing commands, errors may be corrected in the same way as for the standard line editing on the QL. The left and right arrows move the cursor, while in conjunction with <CTRL> they delete characters.

Execution (GO, TRACE and QUICK TRACE)

`G`	GO from current instruction
`G address`	GO from address
`GB address`	GO until breakpoint at address
`GB address condition`	GO until condition at breakpoint
`GR`	GO until return
`Tc`	TRACE one instruction
`Tc number`	TRACE number of instructions
`TUc condition`	TRACE until condition
`TBc address`	TRACE until breakpoint at address
`TBc address condition`	TRACE until condition at breakpoint
`TRC`	TRACE until return
`Q number`	QUICK number of instructions
`QU condition`	QUICK until condition

`QB address`	QUICK until breakpoint at address
`QB address condition`	QUICK until condition at breakpoint
`QR`	QUICK until return
`<ENTER>`	TRACE or QUICK trace

Trace Recall

`RS number`	creates buffer for number of steps
`Rc`	recall last step
`<ENTER>`	recall previous step

Trace Level

`LU`	traces user mode code only, GOes when trace enters a trap (default)
`LS`	traces supervisor mode code as well as user mode code

Breakpoint Control

`B [addresses]`	toggle breakpoint(s) and list them
`BC`	clear all breakpoints

Display Registers or Memory

`Dc [address [number]]`	display memory (in hex and ASCII)
`DIc [address [number]`	display instructions in memory
`DRC`	display registers
`<ENTER>`	continue display

Set Registers or Memory

`SB address byte`	set byte in memory
`SW address word`	set word in memory
`SL address long_word`	set long word in memory
`SDn value`	set data register
`SAn value`	set address register
`SSP value`	set appropriate stack pointer
`SUSP value`	set USP (user stack pointer)
`SSSP value`	set SSP (supervisor stack pointer)
`SSR value`	set SR (status register)
`SPC value`	set PC (program counter)

Modify Memory

`MBc [address]`	modify memory in bytes
`MWc [address]`	modify memory in words
`MLc [address]`	modify memory in long words
`MIc [address]`	modify instructions in memory

Edit Memory

`Ec [address]`	edit memory

Find in Memory

`F value [range]`	find a matching string of bytes.
`F 'string' [range]`	find a matching string of characters

`FI 'string' [range]`	find string in an instruction
`F or FI`	continue search

Open Auxiliary Channels

`Oc name`	opens 'name' as channel 'c'
`Oc`	closes channel 'c'

Calculate Address

`C address`	calculates address and displays it

Macro Command

`CS`	set macro command
`CT`	execute command every trace step
`CB address`	execute command at breakpoint
`CC`	clear CT or CB
`CX`	execute command

2.1 Addresses

Addresses (and most other values) may be given as a simple expression followed by an index. The expression must only include addition and subtraction, and the index may be one or more registers. All 32 bits of a register are used in calculating an address. The are some special addresses which may be used in the expression:

`*`	current PC (address of next instruction)
`L`	address used by the last display command
`N`	default next address
`S`	start address of job

The 'last address' is set by the display (D and DI) commands, modify (MB, MW, ML and MI) commands, and edit (E) command, the 'next address' is set by the find (F and FI) commands as well as by the display (D and DI) and modify (MB, MW, ML and MI) commands.

`L-10`	$10 bytes before the last display address
`N`	the next display address
`*+2`	2 bytes on from PC
`4(A1)`	contents of A1 plus 4
`(A6,D2)`	contents of A6 plus the contents of D2
`100(A6,A1,D2)`	$100 more than A6 plus A1 plus d2

2.2 Conditions

A number of the execution control commands (GO, TRACE and QUICK) use a condition to determine whether to stop execution. The condition is of the form 'register=value' or 'address=value'. The value is assumed to be a word unless it is followed by '.B' for a byte, '.W' for a word or '.L' for a long word. The '=' may be replaced by a '<' for a less than condition, or '>' for a greater than condition. This condition is checked after every instruction during TRACE, or, if there is a temporary breakpoint, the condition is checked at the conditional breakpoint.

2.3 Escape

While tracing and while displaying memory, QMON checks the ESC key. If the key is found to be depressed, the command is terminated.

To ensure that the ESC key is detected, it should be held down until the trace or display stops.

ESC is also used to terminate MODIFY sequences, or to cancel command which has not yet been ENTERed.

3 Invoking QMON

QMON will be most effective if the QL is running in 4 colour mode as it will be possible to display more information than in the 8 colour mode.

QMON is a resident debugger and may be loaded into the QL without having any effect on the operation of the QL. QMON becomes active when invoked from SuperBASIC and remains active until the QL is reset or the job for which QMON was invoked is removed from the QL.

The QMON cartridge or diskette has a boot file and if this is in drive 1 when the QL is reset, QMON will be automatically loaded. Otherwise QMON may loaded by putting the cartridge or diskette in drive 1 and typing:

```
LRUN "FLP1_QMON"   (or LRUN "MDV1_QMON") or LRESPR "QMON_BIN")
```

QMON may now be invoked for job 0 (the SuperBASIC interpreter) by typing:

```
QMON
```

This will produce the prompt 'Qmon>' in window zero. All the QMON commands may now be used. To allow the BASIC interpreter to continue the simple command 'g' (GO) should be used.

```
QMON                    (invoke QMON)
Qmon> D 28000           (display the first few system vars)
28000  D254 0000 0002 8E00  0000 00FC 0002 9800  .T..........
28010  0003 CA00 0003 DC00  0000 0000 0003 DC00  ............
28020  0004 0000 0000 0000  0000 0000 0000 212B  ............
28030  0000 0000 00C0 0001  0000 0000 0000 2CF8  ............
Qmon> G                 (GO)
```

The input and output of QMON will usually share the screen channels of the SuperBASIC interpreter. Other screen channels may be used for both input and output, and other devices (such as a printer or a file on a microdrive or diskette) may be used for displaying memory, or trace output. As the channels are usually shared with SuperBASIC, there may be some conflict. This is reduced by suspending SuperBASIC. SuperBASIC may be released using the normal CTRL SPACE keystroke, and may be suspended with the QMON_W command. The QMON_W command has no other effect.

QMON recognises 4 channels. There is a primary channel which is used for all commands, and it is in this channel that QMON will produce the register display at a breakpoint or other exception. It is also the default channel used for memory displays, etc. Each job monitored

by QMON has its own primary channel. The other three channels (1 to 3) are shared by all jobs being monitored and are used for displaying memory, or for listing the short trace.

QMON by default will trace job 0 (the BASIC interpreter) in channel #0 of the SuperBASIC interpreter. If another job is to be traced, then its default primary channel will be #1.

The primary channel for QMON may be either a normal CONsole window of it may be a special transient window. The transient window appears when QMON is entered to write something to the window, the area of the screen occupied by the window having been saved in the heap. When QMON is left by an execution command (GO or TRACE), then the original contents of the screen are restored. There are 5 five-line transient windows spaced down the screen: window 0 is at the top, while window 4 is at the bottom. In order to be able to get a reasonable amount of information displayed in the transient window, the display mode is set to 512 pixel mode while a transient window is visible. QMON may be invoked for a job already executing in the QL, or it may load and start a job itself.

The command to invoke QMON has a number of forms:

`QMON`	monitor job 0
`QMON [channel] name`	load program 'name' and monitor it
`QMON [channel] number`	invoke or re-enter QMON for job number

The channel may be omitted, in which case, if this is the first time QMON has been invoked for the job, the default primary channel will be used, otherwise the previously used primary channel will be used.

The channel or device must be a CON device. There are three ways in which the channel may be specified:

`#number,`	a SuperBASIC channel number
`name,`	the name of a console device
`number\`	a transient window number

The job number may be found using the JOBS command in the job control extensions provided with QMON. However a good guess would be that the job number will be 1 if it is the only job other than the SuperBASIC interpreter.

3.1 Examples of invoking QMON

`QMON`	monitor job 0 in window #0
`QMON #2,3`	monitor job 3 in window #2
`QMON CON_256x70a0x0,1`	monitor job 1 in a small window
`QMON FLP1_clocks`	set up clocks program and monitor in window #1
`QMON 0\FLP1_clocks`	set up clocks program and monitor in the transient window at the top of the screen

If QMON has already been invoked for a job and that job creates a daughter job, then the daughter job will share the QMON working area with the parent until QMON is invoked for the daughter

4 Concepts

4.1 Trace

Instructions executed by the MC68008 microprocessor in the QL are traced by QMON using the built-in trace facility in the processor. If the trace flag is set, then every time the processor executes an instruction, QMON is called by QDOS. The trace flag is in the status register and is maintained by QDOS for each job. It is therefore possible to trace some jobs in the QL while others continue to run quite normally.

QMON has two trace modes: in the normal trace mode the next instruction to be executed is written to the trace window after every step while in the 'quick trace' mode there is no visible sign that an instruction has been executed. In both modes, however, the conditions that govern the termination of the trace are checked every step. These conditions are one or more of

- a count of instructions executed,
- a check on a register or memory value,
- one or more breakpoints.

4.2 Breakpoints

A breakpoint is an address which is stored in QMON. The job being monitored by QMON will be stopped when the address of the next instruction to be executed is the same as the address of one of the breakpoints.

QMON handles up to six normal breakpoints as well as one special command breakpoint and one temporary breakpoint.

Breakpoints are handled in two ways in QMON. The first way is used with the 'GO' commands. For each breakpoint the first (or only) two bytes of the instruction are saved in the QMON working area for the job, and the illegal instruction '$4AFB' is substituted. To ensure that the first instruction after a 'GO' command is actually executed, even if it is a breakpoint, QMON does one invisible trace step before the breakpoints are actually set.

This mechanism will clearly not work if the code being executed is in read only memory and cannot be changed. The 'TRACE' commands use a different mechanism: the code is not modified to mark breakpoints, but the breakpoint list is checked after every step to see if the address of the next instruction is the same as one of the breakpoints. Whereas setting a breakpoint for a 'GO' command does not influence the speed of execution of a job (until it actually stops!), even 'quick trace' can slow down the execution of a job by a factor of 50.

4.3 Exceptions

The MC68008 processor has two modes of operation, user mode and supervisor mode. Applications programs execute their own instructions in user mode, while 'privileged' code (e.g. the operating system functions) execute in supervisor mode. Code executing in supervisor mode has its own stack (the supervisor stack) and so QDOS extends the concept of privilege to mean that a job executing in supervisor mode cannot be interrupted by the scheduler to allow another job to execute. This means that there need only be one

supervisor stack for all jobs resulting in a considerable reduction in overheads per job by comparison with other multi-tasking operating systems for the MC68000 type of processor.

The mode of operation of the MC68008 is changed to supervisor mode by an exception. Exceptions range from the unpredictable (one of the external interrupts) through the accidental (e.g. illegal instruction) to the controlled (e.g. the trap instructions). QDOS itself is entered by trap instructions and so executes in supervisor mode. QMON is also entered by exceptions and so it, too, executes in supervisor mode. It does not, therefore, use or modify any of the user stack of any job being monitored. However, to allow the QL to continue running other jobs while one is being monitored, QMON reverts to the job's own mode while it is idling waiting for input or output. If QMON is idling in supervisor mode the cursor will not be flashing.

It will not usually be necessary to trace the execution of QDOS traps, so provision is made in QMON to detect a change to supervisor mode during trace and 'GO' automatically. As the status register will now be saved with the trace flag set, when QDOS returns control to the to the application code, the trace will be restored. Unfortunately, the trace will not be activated until one instruction after the trap.

There should rarely be any need to trace supervisor mode code, but if this is to be done then the trace level may be set to supervisor. While QMON is monitoring supervisor mode code, no attempt should be made to display memory or to send trace output to one of the serial ports. The microdrives or floppy disks may, however, still be used. If QMON is being used to trace a QDOS entry, then a GO instruction will GO until the status register is restored on exit from QDOS.

The exception vector used by QDOS during IO subsystem retries is not defined. If you wish to breakpoint or trace the operation of a device driver when handling IO with non-zero timeout, then you should ensure that all jobs executing have had QMON invoked, or, preferably, that the SuperBASIC interpreter is the only job executing.

When QMON is invoked, it creates an exception redirection vector so that QDOS will pass control to QMON when certain exceptions occur. If a job already has an exception vector set up (e.g. to action divide checks) then not all of the QMON exceptions will be redirected. In some cases, if the pointer in the vector does not point directly to an RTE (return from exception) instruction, then the original pointer will be copied into the new vector.

Exceptions not in the following list are irrelevant to the QL and are neither actioned by QDOS nor redirected.

Exception	**Name**	**Action**
Address Error	`Add.er`	always redirected to QMON
Illegal Instruction	`Il.ins`	always redirected to QMON
Zero Divide	`Zero.d`	pointer copied from old vector
CHK Instruction	`Chk`	pointer copied from old vector
TRAPV Instruction	`Trapv`	pointer copied from old vector
Privilege Violation	`Priv.v`	always redirected to QMON
Trace		always redirected to QMON
Level 2 Interrupt		handled by QDOS
Level 7 Interrupt	`Int 7`	always redirected to QMON
Trap #0 to Trap #4		handled by QDOS
Trap #5 to Trap #15	`Trapn`	pointer copied from old vector

When QMON is entered by one of these exceptions, the exception name is written to the primary channel, followed by a register display. If the entry was at a breakpoint, then 'At brp' is written instead of 'Il.ins'

4.4 Trap #4

Execution of a Trap #4 causes problems for a QDOS monitor. If the monitor uses any IO operation after a Trap #4 and before the following #2 or Trap #3, then the action of the Trap #4 will be transferred to the monitor with unpredictable results. For this reason an attempt to trace past a Trap #4 will cause the monitor to enter quick trace mode until the following Trap #2 or Trap #3 has been executed. Ideally the trace level should be set to user. There should be no breakpoints in between the Traps. When single stepping, if the next instruction is a Trap #4, then 'T' should be safe, but 'G' should only be used if it is essential to monitor the execution of the instructions between the Trap #4 and the following Trap #2 or Trap #3.

5 Assembler Format

The assembler and disassembler use Motorola format instructions. The assembler will accept the general form of those instructions which have more than one particular form (e.g. ADD may be used in place of ADDI and ADDA). One limitation is that it is necessary to specify the length of a direct address (e.g. TST.B $280E0.L or MOVE.L $110.W,A2). The disassembler produces instructions in the particular form (e.g. ADDA or ADDI rather than ADD).

The assembler does not accept expressions. Hexadecimal values or addresses should be preceded by $.

6 Command Reference

6.1 GO

`G`	GO from next instruction
`G address`	GO from address
`GB address`	set temporary breakpoint and GO
`GB address condition`	GO until condition at breakpoint
`GR`	GO until return

The GO instructions trace one step invisibly then set $4AFB (illegal instruction) at each breakpoint, clear the trace flag and continue execution of the job being monitored. In all cases execution will cease if QMON is entered by an exception (other than a breakpoint) or at a normal breakpoint. In the case of GB with a condition, if the condition is not met at the temporary breakpoint, then execution will continue.

`G`	GO from next instruction.
`G 3FC50`	set program counter to $3FC50 and GO.
`GB *+6`	set the temporary breakpoint at 6 bytes on from the next instruction and GO from the next instruction.
`GB 3FD46 D1=4`	set the temporary breakpoint at $3FD46 and GO from the next instruction. If the condition (D1.W=4) is not met when the instruction at the temporary breakpoint address is about to be executed, the temporary breakpoint remains set and execution continues. Execution will cease when either the condition is met at the temporary breakpoint address, or QMON is entered at one of the permanent breakpoints.
`GR`	trace one instruction then set the temporary breakpoint at the (return) address to be found on the stack and then continue. If the next instruction is at address $3EFC0 and it is a BSR.L, after this is executed, the address $3EFC4 will be on the stack. Thus the breakpoint is set on the first instruction to be executed after a normal return.

6.2 TRACE

`Tc [number]`	TRACE [number of instructions]
`Q [number]`	QUICK [number of instructions]
`TUc condition`	TRACE until condition
`QU condition`	QUICK until condition
`TBc address`	TRACE until breakpoint
`QB address`	QUICK until breakpoint
`TBc address condition`	TRACE until condition at breakpoint
`QB address condition`	QUICK until condition at breakpoint
`TRc`	TRACE until return
`QR`	QUICK until return

The TRACE instructions set the trace flag and execute the next instruction. If the command was TRACE rather than QUICK and there is more than one instruction to be traced, then the address of next instruction and the instruction itself are written to the channel 'c'. If 'c' is given, it should be in the range 1 to 3. TRACE and QUICK will continue until the trace count is exceeded, the next instruction is at a breakpoint, another exception occurs or the <ESC> is pressed.

`T`	trace one instruction.
`T 20`	trace $20 instructions.
`T2 10`	trace $10 instructions, writing the instructions executed to channel 2.
`QU D0=0`	trace invisibly until the condition (D0.W=0) is met.
`TB 3FD46`	set the temporary breakpoint at $3FD46 and trace until next instruction is at a breakpoint.
`TB3 3FD46 (a1)=20.b`	set the temporary breakpoint at $3FD46 and trace (to channel 3) until the the byte at the address currently in a1 is $20 and the program counter is at the temporary breakpoint.
`QR`	trace one instruction then set the temporary breakpoint at the (return) address to be found on the stack and and quick trace until breakpoint. If the next instruction is at address $3FF30 and it is a BSR.S, after this is executed, the address $3FF32 will be on the stack. Thus the breakpoint is set on the first instruction to be executed after a normal return.

Default Command

If the previous command was a TRACE or GO command, then a blank line (just <ENTER>) is interpreted as trace one instruction.

6.3 RECALL

`RS number`	sets up a buffer for number steps
`Rc`	recalls last step to channel 'c'

RECALL is a backtrace facility which stores the registers for each trace step in a rolling buffer. The number of steps that can be stored depends on the memory available.

`RS`	set up a buffer for 8 steps
`R1`	recall last step to channel 1 ...
`<ENTER>`	... and previous step ...
`<ENTER>`	... and previous to that

6.4 TRACE LEVEL

`LU`	trace user mode code only, GOes when trace enters a trap (default)
`LS`	trace supervisor mode code as well as user mode code

These two commands are used to specify whether QMON will trace the internal operations of QDOS (QDOS code executes in supervisor mode). By default the level is set to user

mode only. However, if an exception occurs which causes QMON to be entered in supervisor mode, then the level is automatically reset to supervisor mode.

Note that, if the level is set to user mode only, then when a trap instruction is traced, the instruction following the trap will not be traced unless it is at a breakpoint.

6.5 BREAKPOINTS

`B [addresses]`	toggle breakpoint(s)
`BC`	clear all breakpoints

QMON can handle up to 7 permanent and one temporary breakpoints. The temporary breakpoints are set by some of the GO and TRACE commands and are cleared on completion of the command. 6 of the permanent set of breakpoints are 'toggled' by the 'B' command, or all of these are cleared by the BC command. At completion of the B command, the current set of permanent breakpoints is listed. 'Toggling' a breakpoint means setting the breakpoint if it is not already set, otherwise clearing it. The seventh permanent breakpoint is set by the CB (macro command on breakpoint) command.

`B`	list the current set of breakpoints.
`B 3ED80`	if there is no breakpoint at $3ED80 set one, otherwise clear it. Then list the current set of breakpoints.
`B 3ED80 3EDF8`	toggle the breakpoints at $3ED80 and $3EDF8.

6.6 DISPLAY

`Dc [address [number]]`	display memory (in hex and ASCII)
`DIc [address [number]]`	display instructions in memory
`DRc`	display registers

The display commands can all send their output to the auxiliary channel 'c'. If given 'c' should be in the range 1 to 3.

The format of these displays are quite different from each other.

The display memory command displays on each line:

- the start address of the line,
- or 16 bytes (depending on the window width) in HEX,
- the same bytes in ASCII if printable or else '.';

while the display instructions command displays on each line:

- the address of the instruction,
- the first 2 bytes of the instruction,
- the disassembled instruction.

The default number of lines for these is 16 or one less than the height of the display window, whichever is less. The default display address is updated to be the address after the end of the display.

The display registers command displays:

- the status register (in hex and the individual flags and the value of the interrupt mask)
- the alternative stack pointer,the values of the
- data registers and the 8 address registers,
- the next instruction in display instruction format.

`D 28000`	display the start of the system vars.
`D1`	display from end of previous display to channel 1.
`D L-10 10`	display $10 bytes previous to last display address
`DI *`	display instructions starting at the next instruction

Default Command

If the previous command was a 'D' or 'DI' command, then a blank line is taken to be another 'D' or 'DI' command to the same channel, and displaying the same number of bytes or instructions, starting from the new default address.

`DI1 3FCE0 8`	displays 8 instructions from address $3FCE0 on channel 1 ....
`<ENTER>`	.... displays the next 8 instructions too.

6.7 SET

`SB address byte`	set byte in memory
`SW address word`	set word in memory
`SL address long_word`	set long word in memory
`SDn value`	set data register
`SAn value`	set address register
`SSP value`	set appropriate stack pointer
`SUSP value`	set USP (user stack pointer)
`SSSP value`	set SSP (supervisor stack pointer)
`SSR value`	set SR (status register)
`SPC value`	set PC (program counter - the address of the next instruction)

The SET commands set a single value.

`SW 3E7C4 40`	sets the word at $3E7C4 to $0040.
`SD4 63`	sets D4 to $00000063.

6.8 MODIFY

`MBc [address]`	modify memory in bytes
`MWc [address]`	modify memory in words
`MLc [address]`	modify memory in long words
`MIc [address]`	modify instructions in memory

The modify commands start a dialogue in either the primary channel window, or in the auxiliary channel 'c' (in the range 1 to 3) which must be a CONsole channel. QMON writes out the address and the value or instruction at that address, and the user can

- press ENTER to leave the value or instruction unchanged,
- press UP ARROW to go back a byte, word (MW or MI) or long word
- press DOWN ARROW to go on a byte, word, long word or instruction
- press ESC to stop the dialogue,
- retype the value or instruction followed by enter
- or edit the instruction using cursor keys in the normal way.

`Qmon> MB 38798`	start modifying bytes at $38798
`38798 70`	ENTER leaves byte unchanged

`38799 1e`	byte changed to $1E
`3879A 72 <ESC>`	ESC exits
`Qmon> MW`	modify words at default address
`3879A 7204 <ESC>`	ESC exits
`Qmon> MI`	modify instructions at default address
`3879A MOVEQ #$4,D1`	ENTER leaves instruction unchanged
`3879C MOVEQ #$FF,D3`	... and so on.

Because the up arrow moves back by one word in the MI command, this will tend to create spurious disassemblies when there are instructions which are more than a word long. Equally, this facility may be used to get the disassembler back into alignment if the MI command does not start on the first word of an instruction.

6.9 EDIT

`Ec [address]`	edit memory in specified channel

This is a hexadecimal and character window based editor. Memory contents may be changed simply by overtyping with the new values. The memory display is similar to that produced by the display command with memory addresses, hexadecimal values and characters. The up, down, left and right keys are used to move the cursor, and <TABULATE> is used to move between the hexadecimal area and the character area. <ESC> exits from the editor.

6.10 FIND

`F value [range]`	find a string of bytes in memory
`F 'string' [range]`	find a string of bytes in memory
`FI 'string' [range]`	find a string within a disassembled instruction
`F or FI`	continue last F or FI

These commands search for strings of bytes or instructions. The default range for the search is from the base of the system variables up to the top of RAM. If a lower limit is given for the search range, then that default is reset, if both a lower limit and an upper limit are given then both defaults are reset. If no parameters are given then the previously specified parameter will be used, and the search will start one byte or one instruction beyond the last match found.

The display address will be set to the even address which comes 8 bytes before the match. This means that a D or DI command will display the context of the match, while a modify command will start well before the match.

The value should be specified in hexadecimal and should have not more than 64 digits (up to 32 bytes). Find string should be specified with a string of not more than 32 characters. Find Instruction should also be specified with a string of not more than 32 characters. The FI command scans the memory disassembling every word, each disassembled instruction is then searched for a matching string. This enables references to particular addresses to be found as well as searching for particular instruction formats. (Note that the disassembler uses hexadecimal notation and the particular forms of commands: ADDA, ADDI, etc.) As Finding an Instruction is very slow, the Find can be interrupted by pressing <ESC>.

`F 4AFB`	find two bytes $4A and $FB in default range.
`F 704774 37000`	reset range from $37000 up to default top, and find 3 bytes $70 $47 $74
`F 'JOBS'`	find the string 'JOBS'
`F`	.... find next occurrence of 'JOBS' ....
`D`	.... and display the memory around it
`FI 'addq.l #4,a7'`	find the instruction ADDQ.L #4,A7

6.11 OPEN

`Oc name`	opens 'name' as the debugger channel 'c'
`Oc`	closes or detaches debugger channel 'c'

The first action of the open command is to close or detach the channel open already as channel 'c'. If the channel is 'owned' by QMON, then the channel is closed, but if the channel is 'owned' by the SuperBASIC interpreter, it is merely forgotten.

If no name is given, no new channel is opened. Otherwise, a new channel is opened to the device or file specified. 'c' must be in the range 1 to 3 and the channel thus opened may be used by trace, display or (if a CONsole) modify commands.

These auxiliary channels are shared by all jobs being monitored by QMON.

When first loaded QMON has channel 1 opened to BASIC #1 and channel 2 opened to BASIC #2.

`O3 MDV2_LOG`	open file MDV2_LOG (a new file) as QMON auxiliary channel 3
`O3`	close or detach auxiliary channel 3

6.12 CALCULATE

`Cc address`	calculates the given address and writes it in hexadecimal and decimal to channel 'c'

This command is used to calculate an address.

`C 10(A6,A1)`	if A6 is $3b668 and A1 is $448 then this will write out: `3BAC0  244416`

6.13 MACRO COMMAND

`CS`	set macro command
`CT`	execute command every trace step
`CB address`	execute command at breakpoint
`CC`	clear CB and CT
`CX`	execute command

A macro command is a compound command which can be invoked directly, at a specified breakpoint or at every trace step. The command is set up by the CS command. It is a single line with one or more commands separated by the '\' symbol. The command may be used to expand the short trace produced by QMON, or to display memory contents or set memory locations or registers at a breakpoint. If a command is to be executed at a breakpoint, then, if execution of the job is required to continue, the last command on the line should be a GO or QUICK instruction as appropriate.

```
Qmon> CS
> D1 (A1) 8 \ SSP -4(SP) \ SL (SP) (D0) \ G
Qmon> CB 36786
Qmon> G
```

Just before the instruction at the breakpoint $36786 is executed, 8 bytes pointed to by A1 will be written to channel 1, then the stack pointer will be decremented by 4 and the contents of D0 put on the stack.

```
Qmon> CS
> D3 (A1) 8 \ D3 38688 8
Qmon> CT
Qmon> TU D6>10
```

Until D6 is greater than 10, at every step there is a partial memory display to channel 3.

7 EXAMPLES OF USE

Note: press the ENTER key at the end of each command.

7.1 SuperBASIC Extensions I

To illustrate the use of QMON while developing extensions to the QL SuperBASIC, the JOBS procedure which is supplied with QMON will be examined.

This first example illustrates the use of QMON in 512 pixel (4 colour) mode. This is the preferred mode for QMON as colour is of little help while the advantage of 80 column output over 40 column is very great. The example SuperBASIC Extensions II illustrates the use of 256 pixel mode.

- RESET the QL,
- put QMON in drive 1 and press F1.

When the cursor appears QMON will be loaded but inactive. Before invoking QMON load the job control extensions and try out the JOBS command by typing

```
LRUN MDV1_JOB              (or LRUN FLP1_JOB)
JOBS
```

The list of jobs currently executing in the QL is written to window #1; there should only be one, the SuperBASIC interpreter, being job 0, tag 0, owner 0, priority 32 and no name. Now invoke QMON by typing

```
QMON
```

A prompt 'Qmon> ' should appear in the command window. QMON is now linked into the SuperBASIC interpreter (and, by implication, linked into any jobs created by the SuperBASIC interpreter) and it is waiting for a command

If the JOBS procedure is to be examined in action, a breakpoint should be set to enter QMON when the procedure is called. The entry point of JOBS will not be at the start of the resident procedure area, but we can find it by examining the procedure definition table which will be near the start of the resident procedure area. Now type

```
Qmon> D 28000                (or d 28000)
```

The base of the system variables area is now displayed in the command window. The address of the base of the resident procedure area is at address 2801C, that is the last two groups of digits on the right hand end of the second line. This address should be 3CA00 on a QL with 128 kbytes of RAM. To display the start of the resident procedures in window #1, type

```
Qmon> D1 3CA00               (or d1 3ca00)
```

Those with expanded memory machines will need to work out their own addresses!

The right hand edge of the display in window #1 is mostly nonsense characters and dots. However, from the third line onwards the words 'AJOB', 'RJOB', 'SPJOB, 'JOBS' are visible. This is the procedure definition table.

To look at the start of the JOBS procedure, the start address of the procedure is found by adding the offset (00A0) which precedes the name 'JOBS' in the table to the address of this offset (3CA28).

```
Qmon> DI2 3CA28+A0
```

The code, now listed in window #2, starts with a branch to a routine to get the channel for the JOBS command, followed by a check on the error return from this subroutine. The register D7 is used as a count of the number of lines written to the output channel, and so has one added to it before the heading line is written out. To trace this code set a breakpoint at the start address 3CA28+A0:

```
Qmon> B L                    (L is the last address used)
BRP 3CAC8                    (confirms breakpoint set)
Qmon> G                      (go on back to SuperBASIC)
JOBS                         (do JOBS procedure)
```

The response to this should be the message 'At brp' (at breakpoint) followed by a display of the registers. The next instruction to be executed (a BSR.L) is displayed at the end. This call to a not very interesting routine is bypassed.

```
Qmon> GR                     (go until return)
```

The Z flag in the condition code register should be set, so that the conditional branch (BNE) should not be taken. The condition code register is the less significant byte of the status register (SR) and is in the first line of the register display. The individual flags X, N, Z, V, and C are put in the line if they are set. The digit is the current interrupt mask value.

```
Qmon> T                      (trace, just <ENTER> would do)
```

One instruction has been traced and the next is displayed. From now on just pressing ENTER will trace one instruction at a time. Note that when the next instruction to be traced is a TRAP then the trace is suspended until the instruction after the TRAP has been executed. As this is is usually a TST.L D0, this is not a very serious problem.

Repeated operations in QMON are interruptable using the ESC key. Type:

```
Qmon> DI2 * FFFF             (display many instructions)
```

pressing ESC will stop the display, pressing ENTER will restart it.

7.2 SuperBASIC Extensions II

This a repeat of the SuperBASIC Extensions I, but for TV mode.

- RESET the QL,
- put QMON in drive 1 and press F2.

When the cursor appears, QMON will be loaded but inactive. Before invoking QMON load the job control extensions and try out the JOBS command by typing

```
LRUN MDV1_JOB              (or LRUN FLP1_JOB)
JOBS
```

The list of jobs currently executing in the QL is written to window #1; there should only be one, the SuperBASIC interpreter, being job 0, tag 0, owner 0, priority 32 and no name. Now invoke QMON window #1 by typing

```
QMON #1
```

A prompt 'Qmon> ' should appear in the window #1. In this mode there is insufficient room for the register display in the command window, so it is necessary to use a larger window for QMON. QMON is now linked into the SuperBASIC interpreter (and, by implication, linked into any jobs created by the SuperBASIC interpreter) and it is waiting for a command.

If the JOBS procedure is to be examined in action, a breakpoint should be set to enter QMON when the procedure is called. The entry point of JOBS will not be at the start of the resident procedure area, but we can find it by examining the procedure definition table which will be near the start of the resident procedure area. Now type

```
Qmon> D 28000 20           (or d 28000 20)
```

The base of the system variables area is now displayed in window #1. The address of the base of the resident procedure area is at address 2801C, that is the last two groups of digits on the right hand end of the fourth line. This address should be 3CA00 on a QL with 128 kbytes of RAM. To display the start of the resident procedures, type

```
Qmon> D 3CA00              (or d 3ca00)
```

The right hand edge of the display in window #1 is mostly nonsense characters and dots. However, from the third line onwards the words 'AJOB', 'RJOB', 'SPJOB, 'JOBS' are visible. This is the procedure definition table.

To look at the start of the JOBS procedure, the start address of the procedure is found by adding the offset (00A0) which precedes the name 'JOBS' in the table to the address of this offset (3CA28).

```
Qmon> DI 3CA28+A0
```

The code, now listed in window #1, starts with a branch to a routine to get the channel for the JOBS command, followed by a check on the error return from this subroutine. The register D7 is used as a count of the number of lines written to the output channel, and so has one added to it before the heading line is written out. To trace this code set a breakpoint at the start address 3CA28+A0:

```
Qmon> B L                 (L is the last address used)
BRP 3CAC8                 (confirms breakpoint set)
Qmon> G                   (go on back to SuperBASIC)
JOBS                      (do JOBS procedure)
```

The response to this should be the message 'At brp' (at breakpoint) followed by a display of the registers. The next instruction to be executed (a BSR.L) is displayed at the end. This call to a not very interesting routine is bypassed

```
Qmon> GR                  (go until return)
```

The Z flag in the condition code register should be set, so that the conditional branch (BNE) should not be taken. The condition code register is the less significant byte of the status register (SR) and is in the first line of the register display. The individual flags X, N, Z, V, and C are put in the line if they are set. The digit is the current interrupt mask value.

```
Qmon> T                   (trace, just <ENTER> would do)
```

One instruction has been traced and the next is displayed. From now on just pressing ENTER will trace one instruction at a time. Note that when the next instruction to be traced is a TRAP then the trace is suspended until the instruction after the TRAP has been executed. As this is is usually a TST.L D0, this is not a very serious problem.

Repeated operations in QMON are interruptable using the ESC key. Type

```
Qmon> DI * FFFF           (display many instructions)
```

Pressing ESC will stop the display, pressing ENTER will restart it. The same principle holds for output to an external device. If a serial printer is available, plug it into SER1, type

```
Qmon> O3 SER1             (open SER1 as channel 3)
Qmon> DI3 * FFFF          (display on printer)
<ESC>                     (stops the printer output)
Qmon> G                   (carry on)
```

7.3 SuperBASIC Extensions III

After trying the previous examples, it is a simple matter to use QMON to generate some trivial code. The first few bytes of the QMON and JOBS extensions are only used for initialisation. Once called they may be overwritten with complete safety. So, to illustrate the use of the QMON assembler type

```
QMON
Qmon> MI 3CA00            (modify instructions)
3CA00 LEA $3CA0E(PC),A1   (first instruction)
```

Type

```
MOVEQ #$F6,D0<ENTER>
```

This instruction is accepted, but is a different length from the previous instruction, and so the disassembler now makes a valiant, if incorrect, attempt at interpreting the next word as the start of a four byte instruction.

Type

```
          RTS<ENTER>
          <ESC>
   Qmon> G
```

The escape returns to QMON command mode. 'G' returns to BASIC.

This code sets the error register to 'end of file' and returns. Try

```
CALL RESPR(0)              (call base of resident procs)
```

The message 'end of file' should be written out.

7.4 Tips

If you are uncertain as to where to put a breakpoint, use the MI command and move through the code using the down arrow (or possibly up arrow) key, and, when you have found the instruction, press <ESC> and then type

```
B N                        (breakpoint at next address)
```

If you are uncertain as to what is about to happen, use Quick trace rather than Going. Then, if anything untoward happens, you can stop it with <ESC>.

It is a good idea, when you are starting to develop software, to scatter a number of TRAP #15 instructions through your code. These will not affect the normal operation of the code, but, if QMON has been invoked, then QMON will be entered at these instructions.

7.5 Executable Programs

A file 'clocks' is included with QMON as an example of an executable program.

Clocks is a digital clock which executes in a default window which is set up to be in the top right hand corner of window #0 for the default monitor mode windows. The clock displays the day of the week, as well as the day, month and time. Both the default window and the characters displayed may be patched.

The characters displayed in the window are selected using a list of bytes. The first byte is the number of bytes in the rest of the list and each of the following bytes selects a character to be written. If the byte is greater than hex 1F then the byte is the 'value' (or 'code') of a character to be written. If the byte is between 0 and $17 (inclusive) then it is a pointer to a buffer containing the characters of the day and date:

```
00      08      10      18
|       |       |       |
day#yyyy mmm dd hh:mm:ss       (the 4th byte is unset)
```

The list for the default display is:

```
14,0,1,2,C,D,E,F,9,A,B,C,10,11,12,13,14,15,16,17,20
```

The addresses which may be patched in the program CLOCKS are:

```
      Address Length  Value   Meaning

      A8      byte    00     border width
      A9      byte    00     border colour
      AA      byte    10     strip colour
```

```
AB      byte    07      ink colour
AC      word    003C    window width
AE      word    0014    window height
B0      word    01C0    X origin
B2      word    00CE    Y origin

B4      bytes           character list
```

The assembler listing of this program is in the file CLOCKS_LIST, it will help to follow the execution of the program if you have a listing of this file handy. To experiment with this program:

- RESET the QL,
- put QMON in drive 1 and press F1,
- type LRUN FLP1_JOB
- type QMON FLP1_CLOCKS

A register display should appear in window #1. The first instruction of CLOCKS has been executed and the next instruction will set A6 to zero. Type

```
Qmon> D                     (display)
```

The start of the program should be displayed, and the name should be visible on the right hand side of the display. Type

```
Qmon> T 20                  (trace 20)
```

When 32 instructions have been traced another register display will be written. Type

```
Qmon> Q FFFF
```

A large number of instructions will now be traced in quick trace mode, pressing <ESC> will stop execution.

To trace the execution from the point where the clock is read, type

```
Qmon> FI 'Q #$13,D0' S    (find MOVEQ #$13,D0 after the start)
3D116
Qmon> MI                  (confirm it)
3D10E MOVEQ #$FF,D1
3D110 MOVEQ #$A,D3
3D112 SUBA.L A1,A1
3D114 TRAP #1
3D116 MOVEQ #$13,D0
3D118 TRAP #1 <ESC>       (escape)
Qmon> B N                 (set breakpoint at next address)
Qmon> G
```

The program should now stop at the breakpoint. Type

```
Qmon> T                   (or just <ENTER>)
```

The time has been fetched in D1, and the next instructions enter the ROM to convert the time to characters.

```
Qmon> T                   (next is JSR (A2))
Qmon> GR                  (go until return)
```

```
Qmon> T
Qmon> T                    (next is JSR (A2))
Qmon> GR                   (go until return)
Qmon> D (a1) 20            (print 20 bytes from (a1))
```

As the next few instructions are traced, it should become apparent that a loop is being executed, writing out individual characters. D6 is being decremented. To speed up the trace, type

```
Qmon> QU D6=1              (quick trace until D6=1)
```

The trace should stop when D6 has been decremented and there is a conditional branch BGT.S as the next instruction. The final iteration round the loop may now be traced one step at a time. Typing the command 'G' will cause the program to execute until it comes to the breakpoint again, so type

```
Qmon> BC                   (clear breakpoints)
Qmon> G                    (carry on)
```

Use CTRL SPACE to release SuperBASIC, (this, and CTRL C, could have been done at any stage to list directories etc.) and type

```
JOBS
```

There should now be two jobs running. The clock is job 1, tag 0 owner 0, priority 1 and name 'Clocks'. To start tracing it again, type

```
   QMON 1                  (monitor job 1)
or QMON 0\1                (monitor job 1 in transient window 0)
```

8 Quick Reference Guide

GO and TRACE

G		GO from current instruction
G	address	GO from address
GB	address	GO until breakpoint at address
GB	address condition	GO until condition at breakpoint
GR		GO until return
T	c	TRACE one instruction
T	c number	TRACE number of instructions
TU	c condition	TRACE until condition
TB	c address	TRACE until breakpoint at address
TB	c address condition	TRACE until condition at breakpoint
TR	c	TRACE until return
Q	number	QUICK number of instructions
QU	condition	QUICK until condition
QB	address	QUICK until breakpoint at address
QB	address condition	QUICK until condition at breakpoint
QR		QUICK until return

RECALL

RS	number	creates buffer for number of steps
R	c	recall last step

TRACE level

LU		traces user mode code only, GOes when trace enters a trap (default)
LS		traces supervisor mode code as well as user mode code

BREAKPOINTS

B	addresses	toggle breakpoint(s) and list them
BC		clear all breakpoints

DISPLAY memory

D	c address number	display memory (in hex and ASCII)
DI	c address number	display instructions in memory
DR	c	display registers

SET memory and registers

SB	address byte	set byte in memory
SW	address word	set word in memory
SL	address long word	set long word in memory
SD	n value	set data register
SA	n value	set address register
SSP	value	set appropriate stack pointer

```
SUSP value                  set USP (user stack pointer)
SSSP value                  set SSP (supervisor stack pointer)
SSR  value                  set SR (status register)
SPC  value                  set PC (program counter)
```

MODIFY memory and registers

```
MB   c address              modify memory in bytes
MW   c address              modify memory in words
ML   c address              modify memory in long words
MI   c address              modify instructions in memory

E    c address              edit memory
```

FIND in memory

```
F    value range            find a matching string of bytes.
F    'string' range         find a matching string of characters
FI   'string' range         find string in an instruction
F or FI                     continue search
```

OPEN and Close

```
O    c name                 opens 'name' as channel 'c'
O    c                      closes channel 'c'
```

CALCULATE address

```
C    address                calculates address and displays it
```

Macro COMMANDS

```
CS                          set macro command
CT                          execute command every trace step
CB   address                execute command at breakpoint
CC                          clear CT or CB
CX                          execute command
```

9 Job Control Extensions

There are four job control extensions in the file JOBS_BIN. These are identical in form to the commands in the Sinclair QL Toolkit.

```
JOBS                                    list all jobs
JOBS #channel                           list all jobs to channel
RJOB job number, job tag, error code    remove job
AJOB job number, job tag, priority      activate job
SPJOB job number, job tag, priority     set job priority
```

The job number and tag are listed with the job name by the JOBS command. A job may only be activated if it has a priority of zero. On activation, a job will start execution at the start address.

10 QMON Version Updates

This version of QMON is the first revision to QMON in more than 5 years. In this time there had been a number of changes the QL world which have left QMON behind. This version is slightly larger, and copes with many of these changes.

10.1 Minerva

The QMON exception handling allows for the Minerva second screen. The QMON SuperBASIC command copes with integer constants.

10.2 Pointer Environment

SuperBASIC is automatically suspended by the QMON command if

a) QMON is invoked for another Job and
b) QMON is invoked in channel owned by SuperBASIC.

The new command QMON_W suspends SuperBASIC - you can still break in with CTRL SPACE.

The QMON output window is automatically picked before it is used.

The Job being monitored is picked when you GO.

Within QMON, the Job being monitored can be picked momentarily:

F1	0.5 seconds
F2	1.0 seconds
F3	2.0 seconds
F4	4.0 seconds
F5	8.0 Seconds

10.3 GOLD card / Atari ST

The keyboard auto-repeat is independent of processor speed.

10.4 General

The Find buffer has been enlarged to 32 bytes

11 General Structure of QMON

QMON divides into four distinct sections plus utilities.

Setup	called from SuperBASIC in QL QMON
Exception handling	vectored entries
Commands	display M/C status, modify M/C status or set QMON parameters
(Dis)assembler	single line assembler/disassembler
QMON utilities	I/O, numeric conversions, etc
SuperBASIC utilities	procedure parameter handling

There are some general rules which apply to the code of QMON. These rules do not limit the generality of QMON, but make it possible for QMON (which is entered on exception and thus uses the supervisor mode stack) to trace jobs in the multitasking environment of QDOS which has a single shared supervisor stack and where supervisor mode code is treated as atomic.

While waiting for I/O, QMON idles in the mode of the job (or task) which caused the entry into QMON.

```
QMON does not use any user mode stack.
```

Any path of subroutine calls that leads to an I/O call, is required to maintain the supervisor stack in a clean state.

```
QMON does not modify the base register A6.
```

11.1 Setup

The setup code is environment specific. In the case of the QL, the setup code is called from SuperBASIC and allocates the QDOS exception vector together with a QMON working area. QMON uses four I/O channels, these are identified by a long word. In the case of QL QMON, this long word is a channel ID.

11.2 Exception processing

The exception processing starts with a vectored jump to a set of branches to subroutines, followed by an exception name. The effect of this is to put a pointer to the exception name on the stack. Illegal instruction is used as a breakpoint, so this is flagged in the MSB of address. TRACE exception is indicated with a zero address.

The first action of the exception processing code is to set the pointer to the QMON working area. In the QL QMON, this is the same as the exception vector address. Next, the registers are saved. (Note that in the QL, a program's data area is potentially moveable; A6 (base register) and USP are liable to be changed whenever I/O is performed.)

If the primary channel save area pointer is set, then the command window is swapped into the screen. This is QL specific.

11.3 Commands

When the exception processing is complete, QMON will either return to the job or task, or call the command routine to accept commands to display or modify memory or registers, or to set the QMON parameters for breakpoints or tracing.

The command routine is also entered directly from SuperBASIC.

The command table is in the main program and may be extended or altered without any difficulty.

11.4 Assembler / disassembler

The assembler and disassembler are two independent modules which share the instruction definition tables.

11.5 QMON utilities

There are three QMON utility routines. One is the QDOS specific I/O routine, the other two are the general purpose routines for getting items from the buffer and putting them into the buffer.

11.6 SuperBASIC utilities

The SuperBASIC utilities are called from the QL setup routine only.

11.7 Impure code

There is only one instance of impure code. This is the auxiliary channel table embedded in the QDOS specific channel switching routine.

11.8 Register usage

D0	(together with the status register) is used for error code returns; also used for loop counters etc.
D1	is used to hold the next character or digit when unbuffering.
D2	is returned from GET with the value of a number or address expression. It also holds the value of the last or only parameter set by COMMAND before a command routine is called.
A0	is a running pointer to the buffer (used by COMMAND, GET, PUT, DIS and SING).
A1	is a pointer to the code to be assembled or disassembled by DIS and SING.
A3	is the address set as the first parameter of two by COMMAND, and then used by the command routines as a pointer to memory.
A5	always points to the QMON data area.
A6	is left alone.

12 QMON and JMON Version 2.06

There are two additional facilities in QMON and a completely independent version of QMON called JMON which is specifically adapted for tracing and debugging jobs and has a new user interface using the Extended Environment.

QMON	is in the file QMON.
JMON	is in the file JMON.

In addition, QMON and JMON now detect the processor type (6800x, 68010, 68020, 68030 and 68040) and adapt their stack frame and cache handling appropriately. The assembler and disassembler are still limited to the 68000 instruction set.

Either QMON or JMON should be loaded as resident extensions (LRESPR or RESPR, LSYTES, CALL). It will not normally be necessary to load both.

12.1 New Facilities in QMON and JMON 2.06

12.1.1 TL Command

The traps #5 to #15 do not now necessarily cause the execution of a job monitored by QMON (JMON) to be halted. The trap number is checked against an internal mask and, if the trap number is higher than the mask level, the execution continues. The trap level is set by the TL command.

TL level	set trap mask to "level"
TL D	set trap mask to $D (13) to ignore traps 14 and 15
The initial trap level is 0	all traps are ignored.

12.1.2 Permanent Breakpoints

Permanent breakpoints may be put into a program by inserting the two words $4AFB (the standard illegal instruction) and $EDBE Enter debugger). On entry to QMON (JMON), the program counter is moved to the instruction after the $EDBE.

12.2 JMON

JMON should only be invoked if both the Pointer Interface and the Window Manager have been installed.

JMON can be invoked from SuperBASIC with the command JMON.

JMON jobnumber or filename	monitor a job or start a new job with JMON
JMON 1	monitor job 1
JMON FRED	execute FRED (data default) under JMON

The job is monitored within the windows of a separate monitoring job. This dependent job may be put to sleep or woken while the monitored job is executing or while it is halted under the control of JMON. Waking the monitoring job halts the monitored job. The monitoring job may also be woken by using the JMON command in SuperSASIC.

JMON jobnumber	wakes the monitor for job "jobnumber"