## MEMORY MAP

000000 – FF7FFF RAM

FFD000 – FFDFFF BOOT FLASH

FFE000 – FFEFFF COMPACT FLASH

FFF000 – FFFFFF 68681 DUART

HE Display help

GO [addr] Go from address in user mode

ST [addr] Step from address in user mode

Execute code as for GO but set a tracepoint. After stepping, press space or enter to step again, press g or G to continue and any other key to exit the mode.

Needs step into/step over functionality.

DR Dump registers

DM [blk] Dump memory block

MM [addr] Modify memory from address

SR [word] Display or update status register

SP [addr] Display or update stack pointer

PC [addr] Display or update program counter

An [addr] Display or update address register n

Dn [data] Display or update data register n

Bn [addr] Display or update breakpoint n

BL List breakpoints

DI [blk] Disassemble block

AS [addr] Assemble into memory

BO Boot CP/M68K

[addr] and [data] are optional 32-bit values and [word] is an optional 16-bit value.

[blk] is an optional memory block definition which may be:

from:to 32-bit addresses of beginning and end of block

from size 32-bit address and 32-bit block size

from 32-bit address with size defaulting to previous value

Continue from end of last block with same size

If the blk argument is missing then continue from the previous location with the same block size.

All numeric inputs are hexadecimal by default or can be explicitly &decimal or $hexadecimal

Modify memory (MM) prompts with the next target address and accepts either a stream of hex digits or text prefixed with a “ character.

Hex data may have spaces after any even digit so that byte, word or long data can be visibly delimited. The spaces are ignored on input.

Text data should not be terminated with a “ character and any following the initial one will be inserted into memory.

Press enter at the prompt to end the command.

MM is the only command that will accept an odd address and it will continue from and odd address if run without an explicit argument.

Any following DM or DI command will round the continuation address to ensure an even value if defaulted.

## Monitor Routines

**GETLINE**

Get zero terminated line into line buffer from user. Maximum 80 chars.

Returns A0 = INBUF and D1 = line length. All other registers preserved.

**GETCH**

Get character from terminal into D0.B. All other registers preserved.

**OUTCH**

Output the character in D0.B. All registers preserved.

**OUTS**

Send the null terminated string at A0 leaving A0 pointing at the next byte after the null. All other registers preserved. . The intention is that a string to be output can have nulls where values are to be inserted and A0 is left ready to continue after each value.

**OUTSLN**

Send the null terminated string at A0 leaving A0 pointing at the next byte after the null.

Then send CRLF. All other registers preserved.

**CRLF**

Output a CR LF pair. All registers preserved.

**OUTSAFE**

Send the null terminated string at A0. Print, at most, D1 characters leave A0 pointing after the last character output

NOTE: A0 will never be left pointing at a terminating NULL and will skip over it if it would otherwise be left there. D1 will be decremented by the number of characters output + 1. All other registers preserved. The intention is that a string to be output can have nulls where values are to be inserted and A0 is left ready to continue after each value.

**PRHEXDIGIT**

Print a single hex digit from the 4 LSBs of D0 - Destroys D0.B. All other registers preserved.

**PRHEX2**

Print a hex byte in D0.B - All registers preserved.

**PRHEX2**

Print a hex word in D0.W - All registers preserved.

**PRHEX6**

Print a hex long word in D0.L - All registers preserved.

**PRHEX8**

Print a hex long word in D0.L - All registers preserved.

**PRBIN**

Print D0.W in binary – All registers preserved

**SPACES**

Print D0.W spaces. Leaves SPACE ($20) in D0

**DUMPREGS**

Dump registers without disassembly. Does not destroy INBUF; all registers preserved.

**DUMPMEM**

Dump memory from A1 for D1 bytes. Leave A1 pointing at next memory location. All other registers preserved.

**PARSENUM**

Parse the numerical value at A0.

&10 = decimal 10

$FF = hex FF

F06A = hex F06A (default)

Result in D1

A0 points at next unparsed character

Numbers can only be terminated by space, (, ], : and end of line (null.) A parse error results in a value of BADNUM in the ERROR byte.

**SKIPWS**

Skip whitespace in null terminated buffer at A0 leaving A0 pointing at the next non-whitespace character. It actually skips anything less than '!' for simplicity. GETLINE will only permit the input of whitespace control characters in any case.

Set Z flag in CCR if A0 is pointing at terminating null on exit.

## Tracing

Tracing must be able to skip over subroutine calls and should always skip over traps and calls into the monitor.

**Skipping over BSR and JSR**

If the next instruction is a BSR or JSR and the user steps over, set the trace trap and set a flag to indicate we are stepping over. Execute, and when the trace exception is caught set an illegal instruction at the return address on the user stack unless there is already a breakpoint there. Then continue normal execution. The ILLEGAL instruction can be used for stepping over subroutines to ensure that all of the BKPT instructions are available to the user.

If the target of a BSR or JSR falls anywhere within the monitor code then it must be automatically stepped over. TRAP instructions should also be skipped over as these have a lower priority than trace.

**JSR modes and sizes**

0000101A 4E90 JSR (A0)

0000101C 4EA8 1234 JSR $1234(A0)

00001020 4EB0 0012 JSR $12(A0,D0)

00001024 4EB8 1234 JSR $1234

00001028 4EB9 1234 5678 JSR $12345678.L

0000102E 4EBA 0204 JSR $1234(PC)

00001032 4EBB 0008 JSR VBASE(PC,D0)

Knowing these sizes allows the ILLEGAL instruction to be placed without inspecting the return address if this turns out to be easier.

## Breakpoints

Breakpoints use the BKPT instruction to permit up to 8 breakpoints to be used simultaneously. Breakpoints are installed when the user code is executed and removed when returning to the supervisor. Breakpoints are not installed at the current PC so that you may continue from a breakpoint correctly. When running from a breakpoint it is necessary to set a trace trap so that the breakpoint can be reinstalled once the PC has moved on. This should not require any special consideration beyond setting a flag to indicate that we wish to continue after the trace exception.

Therefore, the behavior of the trace exception should be;

1. Save registers.
2. Uninstall breakpoints.
3. Check the ‘continue’ flag and GO if set.
4. Otherwise go to the trace repeat code to await user input.

The GO code will install all breakpoints except at the current PC. If there is a breakpoint at the current PC then it will set the ‘continue’ flag and set trace mode before returning to the user’s code. The ILLEGAL instruction used t step over subroutines will also be installed when the breakpoints are installed if required.

Tracing will similarly set all breakpoints not at the current PC but no other special consideration is required.

It must not be possible to set breakpoints in the monitor code nor at the location of an existing breakpoint.