INTRODUCTION

TABLE OF CONTENTS

1.0 GENERAL INFORMATION 1-1
1.1 Command Line Syntax and Line Editing 1-1
1.1.1 Command Line Arguments 1-1
1.1.2 Input/Output Redirection 1-2
1.1.3 Multiple Commands 1-3
1.1.4 Command Line Editing 1-3
1.1.5 Program or command abort 1-4
1.1.6 Command or Batch Files 1-5
1.2 VMEPROM built-in commands 1-6
1.2.1 # - Symbolic Command Name 1-7
1.2.2 AF - APPEND FILE 1-8
1.2.3 AS - LINE ASSEMBLER 1-9
1.2.4 ASSIGN - Assign New Input or Output Ports 1-11
1.2.5 BASE - SET/DISPLAY BASE REGISTER 1-12
1.2.6 BENCH - Built in benchmarks 1-13
1.2.7 BF - Block fill 1-14
1.2.8 BM - Block move 1-15
1.2.9 BP - BAUD PORT 1-16
1.2.10 BR - Set/Display/Delete Breakpoints 1-18
1.2.11 BS - Block search 1-19
1.2.12 BT - Block Test 1-20
1.2.13 BV - Block Verify 1-21
1.2.14 CF - COPY FILE 1-22
1.2.15 COLD - Cold start VMEPROM 1-23
1.2.16 CONFIG - READ HARDWARE CONFIGURATION 1-24
1.2.17 CREATE TASK 1-25
1.2.18 DD - Disk Dump 1-27
1.2.19 DF - DEFINE FILE 1-28
1.2.20 DI - Disassembler 1-29
1.2.21 DL - DELETE FILE 1-30
1.2.22 DN - Display/Change the name of a disk 1-31
1.2.23 DR - Display Processor Registers 1-32
1.2.24 DRF - DISPLAY REGISTERS OF THE 68881/68882. 1-33
1.2.25 DT - DATE AND TIME 1-34
1.2.26 DU - Dump S-record 1-35
1.2.27 ED - VMEPROM screen editor 1-36
1.2.28 ER - LIST ERRORS 1-38
1.2.29 EV - SET/RESET EVENT 1-39
1.2.30 FD - File Dump 1-41
1.2.31 FM - FREE MEMORY 1-42
1.2.32 FRMT - Format floppy or Winchester disk 1-43
1.2.33 FS - FILE SLOT USAGE 1-53
1.2.34 GO - Start user program 1-54
1.2.35 GD - Start user program without breakpoints 1-55
1.2.36 GM - GET MEMORY 1-56
1.2.37 GOTO - GOTO string 1-57
1.2.38 GT - Start user program with temporary breakpoint . . 1-58
1.2.39 HELP - HELP 1-59
1.2.40 IA - IF ALTERED 1-60
1.2.41 ID - SET SYSTEM DATE/TIME 1-61
1.2.42 INIT - Initialize a disk for use with VMEPROM 1-62
1.2.43 INSTALL - INSTALL UARTS OR DISK DRIVER 1-63
1.2.44 KM - KILL MESSAGE 1-67
1.2.45 KT - KILL TASK 1-68
1.2.46 LC - LIST DIRECTORY 1-69
1.2.47 LD - LOAD FILE 1-70
1.2.48 LO - Load S-record 1-71
1.2.49 LS - LIST DIRECTORY 1-72

TABLE OF CONTENTS (cont'd)

1.2.50 LT - LIST TASKS 1-74
1.2.51 LV - DIRECTORY LEVEL 1-75
1.2.52 M - Modify Memory 1-76
1.2.53 MD - Display Memory 1-77
1.2.54 MF - MAKE FILE 1-78
1.2.55 MS - Set memory to constant or string 1-79
1.2.56 PROMPT - CHANGE PROMPT SIGN 1-80
1.2.57 RC - RESET CONSOLE 1-81
1.2.58 RD - RAM DISK 1-82
1.2.59 RM - Modify Processor Registers 1-83
1.2.60 RN - RENAME FILE 1-85
1.2.61 RR2 - EPROM programming 1-86
1.2.62 RS - RESET DISK 1-91
1.2.63 SA - SET FILE ATTRIBUTES 1-92
1.2.64 SF - SHOW FILE 1-94
1.2.65 SM - SEND MESSAGE 1-95
1.2.66 SP - DISK SPACE 1-96
1.2.67 ST - SET TASK TERMINAL TYPE 1-97
1.2.68 SV - Save memory to file 1-99
1.2.69 SY - SYSTEM DISK 1-100
1.2.70 T - Trace program execution 1-101
1.2.71 TC - Set trace count 1-103
1.2.72 TIME - Enable/Disable display of the program run time 1-104
1.2.73 TJ - Trace on change of flow 1-105
1.2.74 TM - TRANSPARENT MODE 1-106
1.2.75 TP - TASK PRIORITY 1-107
1.2.76 UN - CONSOLE UNIT 1-108
1.2.77 ZM - ZERO MEMORY 1-109

The following clarifies some general conventions in more detail than what is done in the VMEPROM User's Manual Chapter 3 titled BUILT-IN COMMANDS.

1. Line Assembler/Disassembler

The line assembler of VMEPROM assumes that all immediate values, addresses and offsets are entered in decimal. So hex values have to be proceeded with a dollar ($) sign. In addition, binary values may be used if proceeded by a percent sign ("%") and octal values if proceeded by an at/around sign ("@"). The disassemblers display all values in hex representation.

The line assembler accepts a pseudo opcode of the form DC.B, DC.W and DC.L to defined constant data storage. The disassembler displays all illegal opcodes as DC.W.

Both the line assembler and disassembler support the opcodes as described in Chapter 4 of the VMEPROM Manual.

2. Most of the VMEPROM commands assume that the parameters are given in hex (without a leading $ sign).

However, some values are assumed in decimal and may only be entered in decimal. These are:

port	VMEPROM port numbers are in the range 0-15 and have to be entered in decimal. The only exception is the BP command which allows the port number to be entered in hex (with a leading $ sign) and decimal.
disk	The disk numbers have to be entered in decimal
level	The directory levels have to be entered in decimal
tasks	The task numbers have to be entered in decimal
task priorities	The task priority has to be entered in decimal
error numbers	The error numbers are displayed in decimal and have to be entered in decimal
event	The event number has to be entered in

decimal

memory size	The memory size (as for the FM, GM and CT commands) has to be entered in decimal

In addition, the benchmark number has to be specified in decimal, while the address parameter of the Bench command is to be given in hex. The INIT Command assumes all values to be decimal and the sector count of the DF command has to be given in decimal.

VMEPROM BUILT-IN COMMANDS

1.0 GENERAL INFORMATION

The VMEPROM command interpreter is a set of resident routines for program debugging and handling of the most common kernel functions.

The command interpreter searches for a given command in the following sequence:

1. Is the command defined in the name table ?
2. Is it a built-in command ?
3. Is the command available as a disk file on the current system disk ?

If a match is found in any of the above steps, the command is executed.

The prompt of VMEPROM is a single question mark, followed by a space ("? ").

1.1 Command Line Syntax and Line Editing

1.1.1 Command Line Arguments

The VMEPROM command interpreter allows several options. In general the complete command line is divided into separate arguments. The arguments must be separated by one or more spaces or a comma. If a null-argument has to be entered, it must be represented by a comma only.

```
    Example: ? PROG ARG1,,ARG3,
```

In this example, the arguments number 2 and 4 are null-arguments.

If any argument is using a comma, space, period or one of the I/O redirection arrows, it has to be put in brackets to suspend the command line interpretation.

```
  Example: ? PROG1 (Hello, world.),(<....>),>TEMP
```

The file TEMP now contains the output of PROG1 which may be:

```
           ? SF TEMP
           ARGUMENT 1 was: Hello, world.
           ARGUMENT 2 was: <....>
           ARGUMENT 3 was:
           ARGUMENT 4 was:
           ARGUMENT 5 was:

           ?
```

1.1.2 Input/Output Redirection

VMEPROM supports simple I/O redirection. The specifiers are the signs '<' for input and '>' for output and may appear at any location in the command line, but must be after the command name. Immediately after the redirection signs, a port number or a filename must be specified.

The Port number may be one of the ports available in the system. It must be specified in a hex, ranging from 1-9 and A-F.

The filenames for I/O redirection may be any file residing on the current disk.

The arguments specifying the I/O redirection are removed from the command line by the command interpreter and do not appear in the user program or the built-in command.

```
   Example: ? PROG <TEMP >3 ARG1,ARG2,ARG3,ARG4
```

In this example, the program PROG is started. It is getting all inputs from the TEMP and all output is redirected to port 3.

The I/O redirection uses the following PDOS functions:

- Input from file uses the assigned console input file mechanism of PDOS.
- Input from port reassigns the input port number (PRT$) in the TCB temporarily.
- Output to file uses the spool file mechanism of PDOS together with the Unit 4 port. So the Unit 4 port shall not be used.
- Output to port reassigns the output port number (U1P$) in the TCB temporarily.

1.1.3 Multiple Commands

VMEPROM allows command lines of up to 78 characters. This command line can contain several different commands. The parsing of the command line is terminated at the first period (".") and the remaining command line is saved to be used later.

```
     Example: ? RM D0 12345678.SM 2,Hello
              ? SM 2,Hello
              ?
```

Be careful when modifying a floating point register from the command line as the decimal point is interpreted as a command line separator. If a floating point register has to be modified, the number must be put in brackets.

```
     Example: ? RM FP0 (12.345).SM 2,Hello
              ? SM 2,Hello
              ?
```

1.1.4 Command Line Editing

The PDOS get line (XGLM) primitive is used to get a command line of up to 78 characters into the command line buffer.

Input is normally in replace mode which means an incoming character replaces the character at the cursor. Various control characters can be used to edit the input line.

The following table summarizes the control characters:

[ESC]	= Cancel current line
[CTRL-C]	= Cancel current line
[CTRL-I]	= Enter insert mode
[CTRL-A]	= Recall last entered line
[CTRL-L]	= Move right 1 character
[CTRL-H]	= Move left 1 character
[CTRL-D]	= Delete character under cursor
[RUBOUT]	= Delete 1 character to the left

A [CTRL-I] changes input from replace to insert mode. The mode returns to replace mode when any other editing control code is entered. Replace mode overwrites the character under the cursor. Insert mode inserts a character at the current cursor position.

In either mode, the cursor need not be at the end of the line when the [CR] is entered. The command line is passed as it appears on the screen.

When a line is accepted, it is copied to another buffer (MPB$) where it can be recalled by using the [CTRL-A] character. A [CR] and [LF] are output to the console followed by the recalled line. The cursor is positioned at the end of the line. This is a circular buffer and commands will rotate through it as they are recalled.

Numeric parameters are entered as signed decimal, hex, or binary numbers. All numbers are converted to two's complement 32-bit integers and range from -2,147,483,648 to 2,147,483,647 (hex $80000000 to $7FFFFFFF). All built-in commands assume that numbers are entered in hex if not noted otherwise.

Decimal numbers must be preceded by an ampersand (&), binary values by a percent sign (%).

(Note: Numbers are not checked for overflow. Hence, 4294967295 is equivalent to -1.)

A line beginning with an '*' is ignored. This is very useful to insert comment lines in command files.

1.1.5 Program or Command Abort

There are two basic methods of aborting a running program or command.

The first one is the ABORT switch on the CPU-board. This switch causes a level 7 interrupt to the processor. If a VMEPROM command was under execution at this time, the message "Abort switch pressed" is displayed and control is transferred back to the command interpreter immediately.

If a user program is running when the ABORT switch is pressed, the current contents of the processor registers are saved and a message along with the processor registers is displayed.

The second method is typing ^C twice on the keyboard. If that happens, VMEPROM will abort the current command or program within 1.28 seconds and control is transferred to the command interpreter. The processor register is not saved by this action. They show the same status as they had before the program was started.

1.1.6 Command or Batch Files

If command or batch files are executed, the parameters from the command line can be used. The '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'. '&#' is replaced with the current task number.

```
     Example: ? SF DOIT
              RM &1 &3
              RM &2 &4

              ? DOIT D0,A1,12345678,1000
              ? RM D0 12345678
              ? RM A1 1000
```

1.2 VMEPROM Built-in Commands

The VMEPROM built-in commands are described in detail in this chapter.

The following general notation is used throughout this document:

- Symbolic representation is put in arrows (i.e. <address> where an absolute address has to be inserted, or <filename> where a filename has to be inserted.
- Optional arguments are in square brackets (i.e. [<option>]). Those arguments must not be specified and have a default value.
- If one argument out of more can be selected, the arguments are separated by a "|" (i.e. [B | W | L] to select Byte, Word or Long Word size).
- If more than one out of many possibilities for an argument has to be selected, these are marked with a "&" sign (i.e. [B|W|L&N&O|E] to select B or W or L together with N and O or E).

Some more hardware related commands may be available. These commands are described in detail in the User's Manual of your particular CPU board.

Most of the VMEPROM commands assume that the parameters are given in hex (without a leading $ sign).

However, some values are assumed in decimal and may only be entered in decimal. These are:

Port	VMEPROM port numbers are in the range 0-15 and have to be entered in decimal. The only exception in the BP command which allows the port number to be entered in hex (with a leading $ sign) and decimal.
Disk	The disk numbers have to be entered in decimal
Level	The directory levels have to be entered in decimal
Tasks	The task numbers have to be entered in decimal
Task Priorities	The task priority has to be entered in decimal
Error Numbers	The error numbers are displayed in decimal and have to be entered in decimal

1.2.1 <u># - Symbolic Command Name</u>

```
Format: #
        # <name>
        # <name>,<command string>
```

The symbolic name command is used to display, delete or define a symbolic name for often used command lines. The first format displays all currently defined names, the second deletes a defined name from the list and the third one defines a new name with the command string. VMEPROM supports up to 5 symbolic names with command lines of up to 40 characters.

```
Example: ? # ASM AS 8000       Define ASM for the command AS
         ? #
         ASM: AS 8000

         ? # D DR              Define D for register display
         ? #
         ASM: AS 8000
         D:   DR

         ? ASM                 Invoke ASM command name

         8000                : NOP
                             : _
```

1.2.2 <u>AF - APPEND FILE</u>

```
Format: AF <file1>,<file2>
```

The APPEND FILE command concatenates two files. The first file <file1> is appended onto the end of file <file2>. The file type attribute of <file1> is transferred to <file2>. The contents of <file1> is not affected by the operation.

A [CTRL-C] interrupts this function on a sector boundary, closes both files, and returns to the monitor. This action is reported by the message '^C'.

The APPEND FILE command uses the assembly primitive XAPF.

Example:

```
?      AF templ,temp2   Append file templ to the end of temp2

?
```

1.2.3 AS - LINE ASSEMBLER

Format: AS <address>

The AS command invokes the line assembler/disassembler of VMEPROM. It can assemble and disassemble all 68000/010 instructions and all the PDOS system calls listed in section 4 of this manual. In addition the 68020/68030 version of VMEPROM can assemble and disassemble all 68020/68030 and 68881/68882 opcodes.

The AS command, when invoked, displays the current address offset and the address within the window. Then the current location is disassembled.

After the prompt on the next line, the user can enter one of the following:

1) A valid 680x0 mnemonic.

2) A '#' sign followed by the new address within the window. This is an absolute address change.

3) A '=' to disassemble the same location again.

4) A '+' or '-' sign followed by the number of bytes the address has to be increased or decreased. This is a relative address change.

5) A '.' to exit the line assembler and return control to the command interpreter.

The line assembler of VMEPROM assumes that all immediate values, addresses and offsets are entered in decimal. So hex values have to be proceeded with a dollar ($) sign. In addition, binary values may be used if proceeded by a percent sign ("%") and octal values if proceeded by an at/around sign ("@"). The disassemblers display all values in hex representation.

The line assembler accepts a pseudo opcode of the form DC.B, DC.W and DC.L to defined constant data storage. The disassembler displays all illegal opcodes as DC.W.

Both the line assembler and disassembler support the opcodes as described in Chapter 4 of the VMEPROM Manual.

Example:

```
?   AS 8800                                     Invoke the line assembler

    8800          : NOP
                  : MOVE.L #$123,D1   New opcode entered
    8806          : NOP
                  : -6                Back six bytes
    8800          : MOVE.L #$123,D1
                  : <cr>              Disassemble next instruction
    8806          : NOP
                  : #8900             Go to absolute address 8900
    8900          : NOP
                  : .                 Back to the command
                                      interpreter

    ? _
```

1.2.4 ASSIGN - Assign New Input or Output Ports

```
Format: ASSIGN <port>
        ASSIGN <port>,<output port>
```

The ASSIGN command has two functions, depending on the command line arguments. If the output port is omitted, ASSIGN sets a new input and output port for the current task. If the output port is specified, the default input/output ports are unchanged, but the alternate output ports of the task are changed. The output port specified must be in the range 1-4.

Example:

```
? ASSIGN 3                VMEPROM now uses port 3 for I/O

? ASSIGN 3,2              Use port 3 as unit 2 port
```

1.2.5 BASE - SET/DISPLAY BASE REGISTER

```
Format:  BASE
         BASE <address>
```

The BASE register in VMEPROM is used to offset all memory accesses into the tasks memory. So all debugging can be done relative to address 0, which is actually the begin address of your tasks memory. This saves a lot of typing and makes sure that no other tasks memory is destroyed by a typing error.

Example:

```
? base<cr>                                Display BASE register
Base = 00000000  : <cr>                   No changes

? base 8000<cr>                           Set BASE register to $8000
? base<cr>                                Display BASE register
Base = 00008000  : <cr>

?M 0<cr>                                  Open address $0 +BASE register
8000+0000   A00E : <cr>
8000+0002   0000 : <cr>
8000+0004   0000 : .

?
```

1.2.6 BENCH - Built-in Benchmarks

```
Format: BENCH
        BENCH <#>,<address>
```

These function can execute one of the built-in benchmarks. If only BENCH is entered, a short descriptions of all benchmarks is displayed on the terminal. A benchmark is executed by entering the number of the benchmark (in decimal) and the address where it shall run in memory (in hex).

The following benchmarks are available:

```
Bench  1:  Decrement long word in memory, 10.000.000 times
Bench  2:  Pseudo DMA 1K bytes, 50.000 times
Bench  3:  Substring character search, 100.000 times, taken from EDN,
           08/08/85
Bench   4:   Bit Test/Set/Reset,  100.000 times,  taken from EDN,08/08/85
Bench  5:  Bit Matrix Transposition, 100.000 times, taken from EDN,
           08/08/85
Bench  6:  Cache test, executes 128K bytes program 1000 times
           CAUTION: This benchmark will destroy 128K bytes memory
Bench  7:  Floating Point - 1.000.000 Additions
Bench  8:  Floating Point - 1.000.000 Sines
Bench  9:  Floating Point - 1.000.000 Multiplications
Bench 10:  100.000 Context switches
Bench 11:  100.000 Set system event
Bench 12:  100.000 Change task priority
Bench 13:  100.000 Send and Receive task message
Bench 14:  100.000 Read system time
```

Example:

```
? bench 1 8000      Execute benchmark #1 at address $8000

Bench  1: Decrement long word in memory, 10.000.000 times
Benchmark time = 0:07.23

?
```

1.2.7 BF - Block Fill

```
Format: BF <begin>,<end>,<value>,[B | W | L]
        BF <begin>,<end>,<pattern>,P
```

This command fills the specified memory area with a constant. The type of the constant is defined by the option and may be a Byte, Word, Long word, or Pattern. A pattern is a ASCII string which is to be put in inverted commas. The maximum length is only restricted by the length of the input line, which may not exceed 78 characters. If the pattern contains argument separators, such as space, comma, or full stop, the pattern has to be put in brackets. If no option is specified, a default of Word is assumed.

Example:

```
? BF 8000 8100 4E71     Fill $8000 to $8100 with $4E71
? MD 8000 20            Display memory to $8000

8000  4E 71 4E 71 4E 71 4E 71  4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq
8010  4E 71 4E 71 4E 71 4E 71  4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq

? BF 8000 8100 ("Hello World") P Fill memory with a pattern

?
```

1.2.8 BM - Block Move

Format: BM <begin>,<end>,<destination>

The BM command copies a memory from one area to another. The areas may be overlapped.

Example:

```
? BM 8000 8080 9000 Copy memory from $8000 to $8080 to $9000
? MD 9000 20        Display memory at $9000

9000  4E 71 4E 71 4E 71 4E 71  4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq
9010  4E 71 4E 71 4E 71 4E 71  4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq

?
```

1.2.9 BP - BAUD PORT

```
Format: BP
        BP <port #>
        BP {-}<port #>,<baud rate>
        BP {-}<port #>,<baud rate>,<type>,<UART base addr>
```

The BAUD PORT command initializes a VMEPROM I/O port and binds a physical UART to a character buffer. The command sets the UART character format, receiver and transmitter baud rates, and enables receiver interrupts.

The first parameter <port #> selects the console port and ranges from 1 to 15. This corresponds to the character input buffers defined in the VMEPROM system RAM (SYRAM). If a minus (-) precedes the port number, then the associated port # is stored in the UNIT 2 (U2P$(A6)) variable.

The receiver and transmitter baud rates are initialized to the same value according to the <baud rate> parameter. The <baud rate> parameter ranges from 0 to 8 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, 110, or 38400. Either parameter type is acceptable.

Baud Rates Allowed:

0 =	19200 baud
1 =	9600 baud
2 =	4800 baud
3 =	2400 baud
4 =	1200 baud
5 =	600 baud
6 =	300 baud
7 =	110 baud
8 =	38400 baud

The <type> and <UART base addr> are optional and are included when binding a logical port to a different UART. For <type> information, refer to the User's Manual of your CPU-board.

The <port #> can also be used to set or reset the port flags.

These are bit positions 8 through 15 of the resulting integer value and are defined to the right. It is recommended that the hex format be used when setting these parameters.

$100 + port	= CtrlS CtrlQ protocol
$200 + port	= Pass control characters
$400 + port	= DTR protocol
$800 + port	= 8-bit character I/O
$1000 + port	= receiver interrupts disable
$2000 + port	= even parity enable
$4000 + port	= clear flag bits

If the BP command has no arguments, then a listing of all currently installed ports is listed to the console. The 'Task' parameter indicates the currently assigned task to that port.

Example:

```
? BP
Port  Type   fwpi8dcs   Base     Baud    task
# 1    1     00001100  FF800000  9600      1

? BP 2,1,1,$FF800200   Initialize the UART
?
```

1.2.10 <u>BR - Set/Display/Delete Breakpoints</u>

```
Format: BR
        BR *
        BR <number>
        BR <number>,<address>
        BR <number>,<address>,<command>
        BR <number>,<address>,<command>,<count>
```

VMEPROM supports a maximum of 10 breakpoints in the range 0-9. The BR command is used to set, display or delete breakpoints.

The first format displays all currently defined breakpoints. The second one deletes all defined breakpoints. The third format is used to define or delete one single breakpoint. If the address field is omitted, the breakpoint with the specified number is deleted, if an address is specified, a breakpoint is either defined at this address, or an existing breakpoint is overwritten.

If a count is specified, the program first stops at the breakpoint when this specification has been achieved. The default value is one.

The default action taken by a breakpoint is a display of the breakpoint number encountered and a display of all processor registers.

So there is a forth option of the command line to change the default behaviour at a breakpoint. The command, which can be specified is executed instead of the display described before. The command may not have any arguments and may have a length of up to 9 characters.

The command may be a symbolic name, one of the built-in commands of VMEPROM or a disk file (command file or program).

Example:

```
? br 0 8020              Define breakpoint 0 at address $8020
? br                     Display breakpoints
Defined Breakpoints:
 B0  8020

?
```

1.2.11 **BS - Block Search**

```
Format: BS <begin>,<end>,[/]<value>[,<option>]
        BS <begin>,<end>,[/]<pattern>,P
```

This command searches the specified memory area for a constant. The type of the constant is defined by the option and may be a Byte, Word, Long word, or Pattern. A pattern is a ASCII string which is to be put in inverted commas. The maximum length is only restricted by the length of the input line, which may not exceed 78 characters. If the pattern contains argument separators, such as space, comma, or full stop, the pattern has to be put in brackets. If not option is specified, a default of Word is assumed.

The value or pattern which has to be searched in memory may be preceded by a "/" to look only for locations not containing the value or pattern.

Example:

```
? bs 8000 8100 /4e71                       Search memory  for "not"
Search:  8020    = 4E70                    value
                                           Found

? bs 8000 8100 4e70
Search:  8020    = 4E70                    Search memory for value.
                                           Found

? bs 8000 8100 ("Hello World") P           Search    memory    for
?                                          pattern.  None found.
```

1.2.12 **BT - Block Test**

```
Format: BT <begin>,<end>
```

The Block Test command performs an in-depth memory test within the specified address limits. The following passes are performed:

1) Byte Pattern Test
2) Word Pattern Test
3) Long Pattern Test
4) Word Shift Test
5) Address Test

If any errors are found they are reported with the type of test which failed, the address and the differing values. In addition the error counter in the task control block (TCB) is incremented.

Example:

```
? bt 200000 300000       Test memory from $2000000 to $300000
?
```

1.2.13 <u>BV - Block Verify</u>

Format: BV <begin>,<end>,<destination>

This command compares two blocks of memory. If the specified blocks are not equal, the different values and the memory location is displayed. In addition the error counter in the task control block (TCB) is incremented.

Example:

```
? bv 8000 8080 8080
Verify:  8021    = 70  80A1    = 71

?
```

1.2.14 <u>CF - COPY FILE</u>

Format: CF <file1>,<file2>

The COPY FILE command copies <file1> into <file2>. The original <file2> is destroyed and replaced by <file1>. The file type attribute of <file1> is transferred to <file2>. <file1> is not affected by the operation.

A [CTRL-C] interrupts this function on a sector boundary, closes both files, and returns to VMEPROM. This action is reported by the message '^C'.

Example:

```
? lc
  test          lv            ls            lc
Number of files: 4            Sectors allocated:  5

? cf test,test1
? lc
  test               lv            ls             lc
test1
Number of files: 5            Sectors allocated:  6

?
```

1.2.15 COLD - Cold Start VMEPROM

Format: COLD

The COLD command is used to reinitialize all VMEPROM variables. It takes the same action as a reset, except that the kernel and all associated tasks are not affected.

Example:
```
? COLD
?
```

1.2.16 CONFIG - READ HARDWARE CONFIGURATION

Format: CONFIG

The CONFIG command searches for the available hardware configuration on the VMEbus. This function is implementation dependant.

For details please refer to the User's Manual of you CPU-board.

If you are using Winchester disks, please make sure that the disk drive is up to speed when the CONFIG command is executed.

The CONFIG command also installs the loadable driver for all boards which are available.

Example:

```
? CONFIG
Disk driver FORCE-ISCSI1 installed
UART FORCE-ISIO1 installed
ISCSI-1 :  1 boards available
 ISIO-1 :  1 boards available

?
```

1.2.17 <u>CREATE TASK</u>

```
Format: CT <command>,<size>,<priority>,<port>
        CT ,<size>,<priority>,<port>
        CT <address>,<size>,<priority>,<port>
```

The CREATE TASK command places a new task entry in the task queue and list of the real-time kernel of VMEPROM. Parameters for the new task include a command line, memory size, task priority, and an I/O port. The new task number is reported after the task is created.

The <command> parameter is the command line for the new task.

The string is passed to the new task via a message buffer and hence cannot exceed 64 characters in length.

Multiple commands and parameters may be passed by using parentheses.

If the first parameter is omitted, then the VMEPROM monitor is
invoked.

If an address is specified instead of <command>, this address is interpreted as the start address of a program in memory. The address must be specified in hexadecimal and must start with a number 0-9 not to conflict with a program name.

The amount of memory for the new task is given by <size> and is in 1 Kbyte increments (although rounded to the next 2 Kbyte boundary). The minimum amount of memory is 8 Kbyte. The system memory bit map is searched for a contiguous block of memory equal to <size>. If the search fails to find a large enough block, then memory is taken from the parent task and allocated to the new task.

The <priority> parameter specifies the new tasks priority. The range of task priority is from 1 to 255 where 255 is the highest priority. The highest priority, ready task always executes. Tasks on the same priority level are scheduled in a round robin fashion.

The <port> parameter assigns an I/O port to the new task. Port 0 is the default and is called the phantom port. On the phantom port, all character outputs and conditional inputs are ignored while requests for character input result in the task aborting with error 86. More than one task may be assigned to an output port. The input port is a unique assignment and cannot be shared with another task. Input ports are allocated on a first come basis. No VMEPROM monitor task with the phantom port (port 0) can be created.

After a task is created, the spawned task number is reported. This number is used in killing the new task.

The values for size, priority and port have to be entered in decimal.

Example:

```
? LT
task   pri  evl/ev2  size     tcb       eom       ports      name
*0/0   64            352    00007000 0008D000    1/1/0/0/0  lt

? CT ,100,64,2
Son task number = 1

? LT
task   pri  evl/ev2  size     tcb       eom       ports      name
*0/0    64            352  00007000 0008D000  1/1/0/0/0   lt
 1/0    64      98    100  0005D000 00076000  2/2/0/0/0

? CT TEST,20,63,0
Son task number = 1

? CT 6100,20,63,0
Son task number = 1

?
```

1.2.18 DD - Disk Dump

```
Format: DD <disk>,<sector>
        DD <disk>,<sector>,<count>
```

The disk dump command displays the raw contents of disk sectors on the terminal. An optional count specifies the number of contiguous sectors to be dumped.

The data is represented in hex and ASCII.

The DD command expects the disk number and the count to be entered in decimal while the sector number is assumed to be in hex.

Example:

```
? dd 0 0 1

Disk # 0   Sector = 0 ($0)

0000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0010  00 00 00 0E 00 00 00 00  00 80 09 20 A5 5A FF FF  ........... .Z..
0020  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0030  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0040  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0050  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0060  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0070  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0080  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0090  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00A0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00B0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00C0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00D0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00E0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00F0  FF FF FF FF FF FF FF FF  FF FF FF FF FF F8 00 00  ................
More (cr) ? <esc>

?
```

1.2.19 DF - DEFINE FILE

```
Format: DF <file{;level}{/disk}>
        DF <file{;level}{/disk}>,<sectors>
```

The DEFINE FILE command creates a new file in a disk directory. <File> specifies the file name, and if included, {;level} the file directory level and {/disk} the disk directory number. Defaults for the latter two parameters are the current level and disk number.

The <sectors> parameter specifies the number of contiguous sectors to allocate to the file. One initial sector is allocated if the <sectors> parameter is not specified. Only contiguous files can be defined. A contiguous file facilitates random access to the file data since VMEPROM can directly position to any byte within the file without following sector links.

If a contiguous file is extended past the original allocation length and a non-contiguous sector is appended to the file, then the contiguous file attribute is deleted.

Therefore, even though contiguous files can be extended, you should allocate enough sectors when the file is first defined to handle all anticipated data. Otherwise, random file access slows down.

The length of a contiguous file is specified in sectors. Each sector contains 252 bytes or characters of data. The file size is given by the number of sectors times 252. The maximum file size is limited by the size of the logical disk.

Example:

```
? DF dfl
? LC

  dfl
Number of files: 1             Sectors allocated:  1

?
```

1.2.20 DI - Disassembler

```
Format: DI <address>
        DI <address>,<count>
```

The DI command causes the disassembler to be invoked and display the mnemonic, starting at the specified address. If count is specified, it is interpreted as the number of lines to display. If count is omitted, a full page is displayed on the terminal and the user is then prompted to continue disassembly (enter <cr>) or to return to the command interpreter (enter any other key).

The disassembler supports all 68000/010 mnemonics. The 68020/68030 version of VMEPROM also supports the 68020/68030 and the 68881/68882 opcodes.

Example:

```
? DI 8000 5

8000   NOP
8002   NOP
8004   NOP
8006   NOP
8008   NOP

?
```

1.2.21 DL - DELETE FILE

```
Format: DL <file>
```

The DELETE FILE command removes from the disk directory the file specified by <file>. All sectors associated with that file are deallocated in the disk's sector bit map and freed for use by other files on the same disk. A file cannot be deleted if it has previously been either delete- or write-protected.

These protection flags must be removed with the 'SA' command before the file can be deleted from the disk.

A sector bit map is maintained by VMEPROM on each disk so that file creation and deletion does not require a disk compaction routine to recover lost disk space.

However, frequent file definitions, deletions, and extensions do create small groups of contiguous sectors which tend to fracture files and make the creation of contiguous files impossible. This is remedied by periodically transferring all files to a newly initialized disk which allocates sectors sequentially for each file.

Example:

```
? lc
  df1            df2            temp           df3            dl1

Number of files: 5              Sectors allocated:  14

? dl temp
? lc
  df1            df2            df3            dl1
Number of files: 4              Sectors allocated:  5

?
```

1.2.22 <u>DN - Display/Change the name of a disk</u>

```
Format: DN
        DN <disk#>
        DN <disk#>,<name>
```

The DN command displays or changes the name of a logical disk. If the disk number is omitted, the current system disk is assumed. If no name is given, the current name is displayed, if a name is specified it is assigned to the disk. The disk name is only for "human" readers and is not used by any of the VMEPROM commands.

Example:

```
? DN 6
Disk 6: VMEPROM DOC

? DN 0 Test disk
? DN 0
Disk 0: Test disk

?
```

1.2.23 <u>DR - Display Processor Registers</u>

Format: DR

The DR command displays all processor registers on the screen. The displayed registers are not the real current processor registers, but those which are kept in memory and loaded to the processor when a program is started. When the execution of a program is terminated (by an XEXT instruction, a trap or a breakpoint or any other exception) the processor registers are saved again and can be displayed by the DR command.

See also: 1.2.24 DRF - Display floating point registers

Note: It is processor dependent as to which registers are to be displayed.

Example:

```
? DR
       0        1        2        3        4        5        6        7
D: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A: 00000000 00000000 00000000 00000000 00000000 00001000 00007000 000767FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0

?
```

1.2.24 DRF - DISPLAY REGISTERS OF THE 68881/68882

Format: DRF

This command displays the registers of the 68881/68882 coprocessor. Like the processor registers, these registers are saved and restored whenever a user program is invoked. This command gives an error if no 68881/68882 coprocessor is installed.

See also: 1.2.23 DR - DISPLAY REGISTERS

Note: This command is only available for 32 bit processors.

Example:

```
? DRF

FP0: 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000
FP4: 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000
```

1.2.25 DT - DATE AND TIME

Format: DT

The DT command outputs the current date and time to the user console. These values can be changed by the ID command.

Example:

```
? DT
16-Mar-88
16:47:38

?
```

1.2.26 DU - Dump S-record

```
Format: DU <begin>,<end>
        DU <begin>,<end>,<command line>
```

This command sends an S-Record to the standard output port. It may be redirected with the usual redirection method.

An optional command line may be specified which is sent via the output port before the S-record starts. This can be used to start a load command on the host system.

The following S-record types are supported:

S1	Start record
S2	Data record, this type is needed if the end address is smaller than $8000.
S3	Data record, this type is used if the end address is bigger than $800000.
S7	End-record for S3 records.
S8	End-record for S2 records.
S9	End-record for S1 records.

The address field of all End-records is 0.

Example:

```
? DU 8000 8020
S0030000FC
S11700004E714E714E714E714E714E714E714E714E714E7172
S10F00144E714E714E714E714E714E7162
S9030000FC

? DU 8000 8020 >2
?
```

1.2.27 ED - VMEPROM Screen Editor

Format: ED <filename>

Example:

```
         3.2.x ED - EDIT

        Format: ED
                ED <filename>
```

The ED command invokes the build in screen editor of the VMEPROM.

An existing file can be specified at the command line and will be loaded when the editor starts.

The size of the editing file depends on the size of the tasking memory where the editor works. The editor always works in the character insert mode with a maximum line size of 79 characters. When the line size is exceeded the cursor automatically wraps to the next line. If there is still space in the edit buffer, a new line will be inserted. The screen holds up to 22 (0-21) text lines. Line 22 is left blank and line 23 is the status line. The status line holds the current cursor position and is used for displaying messages and receiving inputs for some commands.

Note : The ED only can work correctly if the terminal is installed with the 'ST' command.

Editor Commands:

1. Help and Status

<CTRL>A	Display the on-line help screen.
<ESC>A	Display editor status information.

2. Cursor Movement

<CTRL>H	Moves the cursor one character position left but does not wrap to the previous line when the left screen side is reached.
<CTRL>L	Moves the cursor one character position right but does not wrap to the next line when the right screen side is reached.
<CTRL>J	Moves the cursor one line down.
<CTRL>K	Moves the cursor one line up.
<CTRL>B	Moves the cursor to the beginning of the current line.
<CTRL>E	Moves the cursor to the end of the current line.
<CTRL>U	Moves the cursor one page upward and centers the screen.
<CTRL>N	Moves the cursor one page down and centers the screen.
<CTRL>T	Moves the cursor to top of file.
<CTRL>Z	Moves the cursor to end of file and centers the screen.

3. Text editing

<DEL>	Deletes one character left from the current cursor position and wraps to the previous line when reaches the left screen boundary.
<CTRL>D	Deletes one character at the current cursor position and merges the following line to the current when it is pressed at the end of the current line.
<CTRL>O	Deletes the current line.
<CTRL>\	Deletes from the cursor position to the end of the current line including the character at the cursor position.

4. Line Buffer

<ESC>G	Get the current line into the line buffer without changing the current line.
<ESC>S	Swap the line in the line buffer against the current line.
<ESC>I	Insert the line in the line buffer before the current line.

5. Text Pattern search

<CTRL>F	Find a text pattern, center screen and place cursor at the end of the found pattern.
<CTRL>P	Repeat last pattern search.

6. File Operations

<CTRL>G	Get a file from the disk and reinitialize the editor.
<CTRL>W	Write the edit buffer contents to a disk file. An existing file will be overwritten.

7. Other Functions

<CTRL>I	Insert TAB at current cursor position.
<CTRL>]	Set TAB spacing (default is every 8th column).
<CTRL>R	Character repeat function. Allowed keys are any printable character and <DEL>.
<CTRL>V	Restarts the editor. All existing text and initializations are lost.
<ESC>Q	Quits the editor and returns to VMEPROM.

1.2.28 <u>ER - LIST ERRORS</u>

```
Format: ER [-c]
        ER 0 [-c]
        ER <error#>
```

The LIST ERROR command has three functions. The first one, with no argument, displays the number of errors found on one of the following commands:

1) Block Test
2) Block Verify
3) Block Search.

The second format, with the argument "0" resets the above error count to 0.

If the optional parameter [-c] is given when using the first two formats, an execution count will be displayed or reset to zero. The execution count will be incremented before it is displayed.

The third format requires a valid error number as an argument and displays the VMEPROM error message associated with <error#>.

Error numbers range as follows:

VMEPROM errors	1- 49
PDOS errors	50- 99
Disk errors	100-299

Example:

```
? ER
Current error count = 6

? er 0

? er 2
Command line argument error

?er 0 -c

?er -c
 Current error count = 0    Execution count = 1
```

1.2.29 EV - SET/RESET EVENT

```
Format: EV
        EV {-|+}<event>
        EV {-|+},<address>,<bit#>
```

VMEPROM events are set, reset, or listed with the EV command. Both logical and physical events can be accessed with EV. The delayed event queue can also be listed or cleared with the EV command.

If the first parameter is zero, the delay queue is cleared. For accessing a logical event, the event number <event> has to be entered. If <event> is proceeded by a plus (+) sign, the event is set and the old status is returned. If <event> is proceeded by a minus (-) sign, the specified event is cleared and its old status is displayed. For accessing a physical event, the second parameter must be the byte address followed by the bit number (0-7), where bit 7 is the most significant bit of the byte. Physical events are set (+), reset(-) and list(_) in the same way as logical events are accessed. If no special sign is specified, the current status of the event is displayed. If <event> is omitted, a status list of all events in the system and all pending delay events are displayed.

The event number has to be entered in decimal.

Current logical event definitions are as follows:

1-63	= Software events	120	= Level 2 lock
64-80	= Software resetting events	121	= Level 3 lock
81-95	= Output port events	122	= Batch event
96-111	= Input port events	123	= Spooler event
112	= 1/5 second event	124	= Reserved
113	= 1 second event	125	= Reserved
114	= 10 second event	126	= Reserved
115	= 20 second event	127	= Virtual ports
116	= Reserved	128	= Local event
117	= Reserved		
118	= Reserved		
119	= Reserved		

Example:

```
? EV
  00000000  00000000  00000000  0000FE00
  EV 128 : TASK 0  SET DELAY = 43 TICS

? EV 10
  Is  0

? EV +10
  Was 0

? EV -10
  Was 1

? EV 10
  Is  0

? EV +,$10000,1
  Was 0

? EV, $10000,1
  Is 1
```

1.2.30 <u>FD - File Dump</u>

Format: FD <file>

The File Dump command dumps the contents of a file on the terminal.

The file contents is displayed in hex and ASCII representation.

Example:

```
? fd test
0000  54 68 69 73 20 69 73 20  61 20 73 61 6D 70 6C 65  This is a
0010  20 66 69 6C 65 2E 20 49  74 20 77 61 73 20 63 72  sample file.
0020  65 61 74 65 64 20 75 73  69 6E 67 20 74 68 65 20  It was created
0030  4D 46 20 63 6F 6D 61 6E  64 0D 6F 66 20 56 4D 45  using the MF
0040  50 52 4F 4D 2E 0D FF FF  FF FF FF FF FF FF FF FF  command of the
                                                        VMEPROM......

?
```

1.2.31 <u>FM - FREE MEMORY</u>

```
Format: FM
        FM {-}<size>
```

The FREE MEMORY command drops memory from your current task.
If the <size> parameter is positive, then the memory is deallocated and made available to the system for other task usage.
If the <size> parameter is negative, then the memory is simply dropped from the current task and is not recoverable.
The size parameter must be entered in decimal.

Example:

```
? FM
No free memory

? FM 20
20 Kbytes free at address $00071800
```

1.2.32 **FRMT - Format Floppy or Winchester Disk**

Format: FRMT

FRMT - DISK HARDWARE FORMAT

Caution: FRMT may only be run when no other tasks are running. The hardware configuration must be checked before this command can be executed (See CONFIG command).

Description: FRMT allows you to define drives and to format and partition disk drives. VMEPROM supports one floppy and up to three Winchester drives for a maximum of four disk controllers.

When you run this command, you may select a drive to access (i.e. F, F0-F8 for the floppy diskette drives or W, W0-W15, for up to 16 Winchester drives). Enter the drive letters followed by a [CR] to access the drive. Please note that all entries must be in upper case letters. If the drive is undefined, you will be prompted with the drive select byte for the controller.

```
?FRMT

68K VMEPROM FORCE Format  Drive Utility
16/03/88

Possible Disk Controllers in this System
are:

     Controller #1 is not defined
     Controller #2 is a FORCE WFC-1
     Controller #3 is a FORCE ISCSI-1

Drives that are currently defined in system
are:

F0 is controller #3, drive select byte $73
F1 is controller #3, drive select byte $74
W0 is controller #3, drive select byte $00
All not named drivers are undefined.

   Select Menu:  W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
  Select Drive:  _
```

If you select either a floppy drive or a Winchester drive that is already defined, FRMT directly enters the Drive Command Menu. If you are installing a new Winchester drive which is currently undefined, then you must enter the controller number and drive select jumpering (0-3). The Drive Command Menu tells you which drive you are currently dealing with and has the following commands:

```
 Select Menu :  W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
Select Drive :  W0[CR]
W0 Main Menu :  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
     Command :  [CR]

 Winchester Drive 0 Menu:

        1) Display/Alter drive Parameters.
        2) Display/Alter Bad Track List.
        3) Format tracks.
        4) Verify tracks.
        5) Display/Alter VMEPROM Disk Partitions.
        6) Write out Header info to disk.
        P) Toggle Unit 2.
        Q) Quit & Select another Drive.

 W0 Main Menu:

        1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
        Command: _
```

When dealing with a floppy drive, the display/alter commands do not allow you to alter the drive parameters, the bad track table, or the disk partitions, and you may not write out header information to a floppy disk. To exit to VMEPROM, you must first return to the Select Drive Menu with the Q) command. Following is a detailed description of the Drive Command Menu commands:

1) Display/Alter Drive Parameters:

The Display/Alter Drive Parameters menu allows you to D)isplay the currently defined drive parameters, A)lter them, R)ead them in from a file, or Q)uit and exit to the Select Drive Menu:

```
W0 Parameters Menu :  A)lter, D)isplay, R)ead file, Q)uit
           Command :  _
```

To display the current drive parameters on a Winchester, enter the 'D' command. The parameters are displayed to the screen. The Drive Parameters that are displayed, and that can be altered are:

```
       Current (type) Drive N Parameters:
                      # of Heads = Number of heads on drive
                  # of Cylinders = Number of cylinders on drive
       Physical Blocks per Track = Actual blocks on a track
        Physical Bytes per Block = Actual bytes per physical block
               Shipping Cylinder = Where to position head before
                                   moving drive
                       Step rate = Controller dependent definition
       Reduced write current cyl = Cylinder to apply reduced
                                   write current
         Write Precompensate cyl = Cyl to apply write
                                   precompensation
```

To alter them, enter the 'A' command. In the alter mode, you enter either: 1) a carriage return to leave the parameter the same and go to the next prompt; 2) a number and a carriage return to change the parameter and go to the previous parameter prompt. The Drive Parameters are displayed one at a time for you to either alter or leave alone.

If you have previously saved out the drive parameters to a disk file, you can restore them by entering the 'R' command, followed by the name of the file. This file may be created using the F)ile command of Drive Command Menu option 6) Write to disk, or it can be created with a VMEPROM editor. The format and syntax of the parameter file is discussed under option 6). Reading this information destroys all other information; replaces the parameters, the bad track table, and the partition definitions.

The 'Q' command returns you to the Drive Command Menu.

For example, look at floppy drive F0 parameters:

```
 Select Menu :  W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
Select Drive :  F0[CR]
F0 Main Menu :  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
     Command :  1[CR]

Current Floppy Drive 0 Parameters:
                      # of Heads = 2
                  # of Cylinders = 80
       Physical Blocks per Track = 16
        Physical Bytes per Block = 256
               Shipping Cylinder = 0
                       Step rate = 0
       Reduced write current cyl = 0
         Write Precompensate cyl = 0

   F0 Main Menu:
        1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
        Command: _
```

As another example, select the W0 Winchester and display the current parameters:

```
   W0 Main Menu:
          1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
        Command:  1[CR]
  W0 Parameters: A)lter, D)isplay, R)ead file, Q)uit
        Command:  D[CR]

  Current Winch Drive 0 Parameters:
                      # of Heads = 16
                  # of Cylinders = 1000
       Physical Blocks per Track = 32
        Physical Bytes per Block = 256
               Shipping Cylinder = 0
                       Step rate = 0
       Reduced write current cyl = 0
         Write Precompensate cyl = 0

    W0 Parameters Menu:  A)lter, D)isplay, R)ead file, Q)uit
               Command:  Q[CR]
          W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ
                         P)Togl Q)Quit
               Command:  _
```

2) Display/Alter Bad Track List:

The Display/Alter Bad Track menu allows you to D)isplay the currently defined bad tracks on the drive (if any), add or delete tracks, C)lear the bad track table, get a H)elp message, or Q)uit and exit to the Drive Command Menu:

```
W0 Bad Tracks Menu:  Bad Track, D)isplay, C)lear, H)elp, Q)uit
           Command:  _
```

To display the current bad tracks on a Winchester, enter the 'D' command. The tracks are displayed on the screen in ascending order as a physical track number followed by the head and cylinder number, separated by a comma and enclosed in parentheses.

To add a bad track to the list, enter either the actual physical track number and a carriage return, or the head and cylinder number desired, separated by a comma and followed by a carriage return. To delete a track, precede the track or head number with a minus sign (-).

Sometimes the bad track table may be incorrect or spoiled. You can start all over again by entering the C)lear table command. The 'Q' command returns you to the Drive Command Menu. In case you have added or deleted some bad tracks, FRMT asks if you want to recalculate the disk partitions on the drive before returning to the drive menu. By altering the number of bad tracks, you also alter the number of logical tracks available for VMEPROM disk partitions. Answer 'Y' or 'N' to the query, as you like.

Note that the SCSI Winchesters handle bad blocks internally. So when you are using the ISCSI-1 controller, the bad blocks defined by the manufacturer are already spared on the disk.

For example, look at the Winchester drive 0 bad track list:

```
    W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl
               Q)Quit
         Command:  2[CR]
W0 Bad Tracks Menu:  Bad Track, D)isplay, C)lear, H)elp, Q)uit
         Command:  D[CR]

 Current Winch Drive 0 Bad Tracks:
231(0,77)    613(1,204)    697(1,232)    700(1,233)    703(1,234)

 W0 Bad Tracks Menu:  Bad Track, D)isplay, C)lear, H)elp, Q)uit
           Command:  Q[CR]
      W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ
                     P)Togl Q)Quit
           Command:  _
```

3) Format Drive/Tracks:

```
Sector Interleave = {default from MCONTB table is listed}
Physical Tracks to Format = {[CR] for beg,end tracks listed}
Ready to Format Drive 0 ? {'Y' or 'N'}
```

This routine first calls the INFMT routine which sets up the format. Then F)ormat makes one or more calls to the TKFMT routine until all the specified tracks are formatted. Between calls, a check for user break ([CTRL-C]) is made, and the track number just formatted is printed to the terminal. If there are errors, you can select either R)etry, Y)es-- add the track to the bad track list, or N)o -- ignore the error and go on.

For example, format a floppy disk with the default sector interleave, 5, and do tracks 0 to 159, inclusive:

```
        Sector Interleave = 5[CR]
        Physical Tracks to Format = 0,159[CR]
        Ready to FORMAT Floppy Drive 0 ? Y[CR]
        Sector Interleave Table:
        1,9,4,12,7,15,2,10,5,13,8,16,3,11,6,14

        Issuing Format Drive Command

        FORMAT Successful!
```

Note that the interleave is "Don't care" for SCSI Winchester drives.

4) Verify Tracks:

```
Physical Tracks to Verify = {default from last format
command}
                          Ready ? {'Y' or 'N'}
```

This routine, after calling INFMT, reads every sector on each track specified. Errors are reported to the terminal. Between calls a check for user break ([CTRL-C]) is made, and the track just verified is printed to the terminal. If there are errors, you can select either R)etry, Y)es -- add the track to the bad track list, or N)o -- ignore the error and go on.

5) Display/Alter Disk Partitions:

The Display/Alter Partitions menu allows you to D)isplay the currently defined disk partitions, A)lter them, R)ecalculate them from the current values, or Q)uit and exit to the Drive Command Menu:

```
  W0 Partitions Menu:  A)lter, D)isplay, R)ecalc, Q)uit
             Command:  _
```

To display the current disk partitions on a Winchester, enter the 'D' command. The partitions are displayed on the screen. The Disk Partitions that are displayed are based on a few parameters, which you can change:

# of Large partitions =	How many large divisions on the drive
# of Floppy partitions =	How many small divisions on the drive
First track for VMEPROM Parts =	Where to begin the disk partitions
Last track for VMEPROM Parts =	Where to end the disk partitions
First VMEPROM disk # =	What is first VMEPROM disk # of partitions

To alter them, enter the 'A' command. In the alter mode, you enter either: 1) a carriage return to leave the parameter the same and go to the next prompt; 2) a number and a carriage return to change the parameter and go to the next prompt; or 3) an escape to go to the previous parameter prompt. The disk partitions parameters are displayed one at a time for you to either alter or leave alone. If you alter the number of disks or the tracks for partitions, then you are asked if you would like to recalculate the partitions. Enter either 'Y' or 'N'. If you only change the beginning VMEPROM disk number then only the disk numbers are reassigned, leaving the base and top tracks of the partitions alone.

You can make the partition information consistent by simply entering the 'R' command. This recalculates the drive partition information using the current values of drive parameters, bad track table, and partition parameters. The 'Q' command returns you to the Drive Command Menu.

```
     W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl
                    Q)Quit
          Command:  5[CR]
W0 Partitions Menu:  A)lter, D)isplay, R)ecalc, Q)uit
          Command:  D[CR]

      Current Winch Drive 1 Disk Partitions:
         # of Large Partitions = 10
        # of Floppy Partitions = 12
  First track for VMEPROM Parts = 0
   Last track for VMEPROM Parts = 15979
          First VMEPROM disk # = 2
     Total # of Logical Tracks = 16000

     Disk #     Logical Trks  Physical Trks   VMEPROM sectors
                Base,Top      Base,Top        Total/{boot}
         2         0,1499        0,1500          47968/47776
         3         1500,2999     1501,3000       47968/47776
         4         3000,4499     3001,4500       47968/47776
         .         .             .               .
         .         .             .               .
         .         .             .               .
        24         15880,15959   15897,15979     2528/2336

     W0 Partitions Menu:       A)lter, D)isplay, R)ecalc, Q)uit
        Command:               Q[CR]
         W0 Main Menu:         1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ
                               P)Togl Q)Quit
                 Command:      _
```

6) Write Header Information to Drive:

The Write Header Information to Drive menu allows you to 1) 'Y' write the information to the drive header, 2) 'N' abort the command and return to the Drive Command Menu, or 3) 'F' write drive information to a file. After assigning the correct parameters to a drive, entering the bad tracks, formatting it, and partitioning it into VMEPROM disk numbers, you still need to write this information to the drive's header. This information must reside on the disk and is used by for the BOOT ROMs and by VMEPROM.

This routine calls the INFMT subroutine to initialize the controller for the new number of heads and cylinders, and then calls the WTHED subroutine which writes out the drive data block (DDB) to the correct sector on the drive, usually sector 0.

To write this information to the drive, enter the 'Y' command. If you have second thoughts, enter the 'N' command.

You should save the information out to a floppy disk file for each Winchester drive. This file makes recovering from Winchester disasters easier. You can either select to only write out the file with the 'F' command, or write the file out after writing out the header information to the drive.

The file syntax is that:

1) lines starting with an asterisk (*) are ignored as comments;
2) parameter key words are four characters long and appear as the first four characters of the line;
3) key words are followed by an equal sign (=) and the value (hex must be preceded with dollar sign ($));
4) bad tracks use the key word TRACK, are followed by an equals sign, and are designated by either the track number or the head and cylinder numbers (separated by a comma);
5) order of the key words is not significant, except that the HEDS definition must precede any TRACK specification using the head, cylinder format; and
6) any unspecified key word parameters are reset to system defaults, and not left as previously entered values.

The drive parameter key words are defined as follows:

HEDS	= # of Heads
CYLS	= # of Cylinders
BPTK	= Physical Blocks per Track
BPBK	= Physical Bytes per Block
SHIP	= Shipping Cylinder
STEP	= Step Rate
REDU	= Reduced Write Current Cylinder
WRTP	= Write Precompensate Cylinder

The disk partition key words are defined as follows:

WPRT	= # of Large Partitions
FPRT	= # of Floppy Partitions
BTRK	= First Track for VMEPROM Parts
ETRK	= Last Track for VMEPROM Parts
BDKN	= First VMEPROM disk #

While reading in the file using the R)ead command of the 1)Parameter menu, FRMT outputs a 'Found:' message, followed by the parameter value when a successful key word match and number conversion is made. This indicates that the parameter was loaded. If a key word match is not made or if the conversion fails, FRMT echoes the line to the terminal preceded by two question marks (??). This indicates that the parameter was not loaded.

Q) Select Another Drive:

If you were working with a floppy drive, the Q)uit command simply returns you to the Drive Select Menu. If you were working with a Winchester, then the Q)uit command asks whether or not to write the new drive data block down to low parameter RAM. Enter either 'Y' or 'N' to this query. If you answer 'N', your configuring session will be lost. It then exits to the Drive Select Menu.

```
W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl
               Q)Quit
  Command:     Q[CR]

  Exit to Select Drive.  Update Param RAM (Y/N) ? Y[CR]
  System Parameter RAM Updated!!
  Select Menu: W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
       Select Drive:  Q[CR]

?
```

1.2.33 FS - FILE SLOT USAGE

Format: FS

The FILE SLOT USAGE command lists all files currently open along with file slot information. When the first file is opened, it is assigned slot number 64; as successive files are opened, they are assigned file slots in numerical sequence down to 1. (Read Only Open allocates slots in the opposite order, from 1 to 32.) The file slot maintains information such as the current file pointers and sector indexes.

This data is defined as follows:

Slot	File slot #
Name	File name;level/disk #
ST	File status
SM	Current sector in memory
PT	Current file pointer
SI	Sector index of SM
EOF	Sector index/# of bytes in END-OF-FILE sector
TN	Lock Task/Open Task
BF	Buffer pointer
FLGS	Lock flag/# Shared

File status is defined as:

ST =	$8xxx	Sector altered	$xx80	Altered
	$4xxx	File altered	$xx04	Contiguous file
	$1xxx	Driver in channel	$xx02	Delete protect
	$xAxx	Read only access	$xx01	Write protect
	$x6xx	Shared random access		
	$x2xx	Random access		
	$x1xx	Sequential access		

Example:

```
? FS
Slot  Name        ST   SM   PT       SI   EOF     TN   BF       FLGS
64    fs1;101/6   C104 0142 00003916 0000 0000/82 0000 0000389E 00000000
```

1.2.34 GO - Start User Program

Format: G
        G <address>
        GO
        GO <address>

A user program in memory is started with this command. The start address may be specified on the command line, or the value of the program counter, as displayed by the DR command, is taken if this field is omitted.

The following actions are taken by VMEPROM if this command is specified:

1) The processor registers are loaded with the user values.
2) The first instruction is executed.
3) If any breakpoints are defined, they are inserted in the user program.
4) The program is continued at the second instruction.

Example:

```
? G 8000
>>> This is a Test <<<

?
```

1.2.35 <u>GD - Start User Program Without Breakpoints</u>

```
Format: GD
        GD <address>
```

The GD command takes the same actions as the G or GO command, except that defined breakpoints are ignored and not inserted in the user program.

Example:

```
? GD 8000
>>> This is a Test <<<

?
```

1.2.36 <u>GM - GET MEMORY</u>

```
Format: GM
        GM <size>
```

The GM command adds memory to the current task. The amount of memory is specified by <size>. The <size> parameter has to be given in decimal. If no parameter follows GM, then all of the available memory is added. No error is reported if the memory request cannot be met.

Example:

```
? FM
No free memory

? FM 20
20 Kbytes free at address $00071800

? GM
? FM
No free memory

?
```

1.2.37 GOTO - GOTO String

Format: GOTO <string>

The GOTO command is used in procedure files to selectively process different commands. When the GOTO command is executed, the procedure file is rewound and all command line entries are ignored until a match is found with the <string> parameter and the command line. All preceding command lines to the match, including the matching command line, are ignored.

Execution continues with the next line.

The console echo flag (ECF$) is set to disable all console output until a match is found or the procedure file is exited. It is again restored after the label is found. Labels beginning with an asterisk are recommended since the monitor ignores them.

Example:

```
? TEST <cr>
? *START
? BT 100000 300000
? ER
Current error count = 0
? GOTO *START
```

1.2.38 GT - Start User Program with Temporary Breakpoint

```
Format: GT <breakpoint>
        GT <breakpoint>,<address>,<command>
```

This is almost the same function as the G or GO command, except that an additional temporary breakpoint is inserted. This breakpoint is automatically removed if the PC reaches this breakpoint. If a command is given, it will be executed at the breakpoint.

Example:

```
 ? GT 8020 8000
 At temporary breakpoint

      0        1        2        3        4        5        6        7
D: 00000000 00000000 00000000 00000100 000066DC 00000000 00000006 00000000
A: 00006290 000766FC 00005060 00006297 000766F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0

?
```

1.2.39 HELP - HELP

Format: HELP
        HELP <command>

The HELP command first displays a short description of all VMEPROM built-in commands on the terminal. Then a more detailed description of all commands is displayed.

After every screen full, the output stops. It may be continued by entering a <cr>. Control is transferred back to the command interpreter on any key other than <cr>.

If HELP is followed by a command name, a short description of this command is displayed.

If HELP is followed by one or more characters, but not a complete command name, a start description of all commands matching with the given character is displayed.

Example:

```
? HE M
M <address> [,B|W|L&N&O|E]    Modify memory contents

?
```

1.2.40 IA - IF ALTERED

Format: IA <file name>.<command>

The IF ALTERED command tests and clears the altered file bit of the directory entry specified by <file name>. If the file had the alter bit set (indicated in the directory listing by a '+' under type), then execution of the command line continues. Otherwise, the rest of the line is ignored.

This command is useful in assembly procedures to update object modules when many files are involved and only a few may have changed.

Example:

```
? IA test.DT
? DT
16-Mar-88
16:47:38

? IA test.DT
?
```

1.2.41 <u>ID - SET SYSTEM DATE/TIME</u>

Format: ID

The SET SYSTEM DATE/TIME command displays the VMEPROM header and prompts for the date and time. The header shows the version of VMEPROM and the used CPU-type as displayed after reset.

The date can be entered in either a day, ASCII month, year form or numeric month, day, year.

Any delimiter can be used to separate date and time parameters.
Pressing [CR] leaves the old date and time.

Example:

```
? ID

************************************************************
*                     V M E P R O M                        *
*                                                          *
*          SYS68K/CPU-20/21   Version 2.0  16-MAR-88       *
*                                                          *
*          (c) FORCE Computers  and  Eyring Research       *
*                                                          *
************************************************************

Date=16-Mar-88
Time=16:46:49
```

1.2.42 <u>INIT - Initialize a Disk for Use with VMEPROM</u>

```
Format: INIT
        INIT <disk>,<directory size>,<disk size>,<disk name>
```

The INIT command initializes a floppy or Winchester for the usage with VMEPROM. The disk must be formatted (see FRMT command).

The required parameters are:

1. disk number
2. number of directory entries
3. physical size of the disk in number of 256-byte sectors
4. disk name

All parameters may be specified on the command line or may be entered interactively after the function has been invoked.

Typical values for INIT are:

```
   for floppies:      128  directory entries
                     2528  sectors

for Winchesters:     512 or 1024 directory entries
                     dependant on the specification with FRMT,
                     option 5.
                     (see FRMT command for details).
```

Example:

```
? INIT 9,128,2336,Diskname
Init:  Disk # 9
       Directory entries: 128
       Number of sectors: 2336
       Disk name: Diskname

?
```

1.2.43 INSTALL - INSTALL UARTS OR DISK DRIVER

```
FORMAT: INSTALL [?]
        INSTALL -<U<type>|W<number>>
        INSTALL U<uart type>,<address | filename>[,board base
                address]
        INSTALL W,<address | filename>[,board base address]
                [,number of desired disks(F/W)|<P>partition
                list][,partition offset]
```

The INSTALL command installs, lists or removes device codes (disk drivers, UARTs). If there is no parameter given, all installed UARTs and disk drivers are shown. If the first parameter is equal to a question mark, all UARTs and disk drivers, which are already in EPROM, are listed.

INSTALL UARTs:

VMEPROM can handle up to eight UART types. Each type has a table of short branches (DSR table) for various subroutines to get, put, baud etc. If a certain UART type is not used in the system, a "NE" status is returned for all calls. To install a new UART type, set the first parameter to U1, U2 up to Un, where n is the number of UART types in the system. If the number is out of range, then an error appears. The ability to pick a type is not currently used in the system. This means that uninstall must first be used with this UART type by preceding the first parameter with a minus sign.
The second parameter can have the filename of the DSR object code or the base address where the object code starts. In case of a filename being written, the INSTALL facility first loads the object code into memory and preserves that memory. It then calls the initialization routine for the card and enters a jump table for this UART into a global jump table for UARTs. In this case, the UART type ALSO reserved a small RAM area of maximum 64 bytes.

The optional third parameter, <board base address>, is the base address of the first card of the new type, as jumpered in the system. The DSR table has the same entries as the standard PDOS UART type, with the following additions. The data word just after the DSR table must contain the characters "U0"(Uzero), the word just after that must have a BRA.S INIT branch to the card initialization routine. The INSTALL assumes that after the initialize call that there is a string, null terminated, which describes the UART type.

If VMEPROM finds the magic word $A557 there after, an uninstall will be supported.

Each UART entry is defined as follows:

```
UDSR    BRA.S   UDG              ;GET CHARACTER
        BRA.S   UDP              ;PUT CHARACTER
        BRA.S   UDB              ;BAUD UART
        BRA.S   UDR              ;RESET UART
        BRA.S   UDS              ;READ UART STATUS
        BRA.S   UHW              ;HIGH WATER
        BRA.S   ULW              ;LOW WATER
        DC.B    'U0'             ;UART ID
        BRA.S   UDI              ;INSTALL
UNAME   DS.B    'NAME',0         ;NAME OF DRIVER (ZERO TERMINATED)
        EVEN
        DC.W    $A557            ;MAGIC
        DC.W    P_TYP            ;PROCESSOR TYP
        BRA.W   UNINS            ;UNINSTALL

        P_TYP  = %000000000000xxxx
                              \\\\__ 68000
                              \\\__ 68010
                               \\__ 68020
                                \__ 68030
```

The INIT call is made by INSTALL in supervisor mode. This routine has the following inputs and outputs:

```
UDI - INSTALL DRIVER
      IN: A1.L = K1$BEGN
          A2.L = OPTIONAL CARD BASE ADDRESS OR ZERO
          A5.L = SYRAM BASE
          A6.L = BEGIN OF TCB
        (A7)   = RETURN ADDRESS
       4(A7)   = RAM ADDRESS IN GLOBAL DSR TABLE
     OUT: D0.W = -1 ERROR
                 NUMBER OF CARDS
```

VMEPROM also supports an uninstall routine with following inputs:

```
UNINS - UNINSTALL DRIVER
        IN: (A7) = RETURN ADDRESS
           4(A7) = RAM ADDRESS IN DSRTAB
```

INSTALL DISK DRIVER:

VMEPROM handles up to four disk drivers linked in a driver list. To install a new disk driver set the first parameter to W. If the device code is resident any where in memory or EPROM, the second assignment <address|filename> should be the start address of the driver. If a filename is given, the INSTALL facility first allocates memory and loads the object code into memory. Then INSTALL calls the initialization routine (INIT) for the disk controller and enter the new disk driver into the driver list. If there are already four disk driver installed an error will occur and you first had to uninstall any driver by setting the first parameter to -Wn. n is the number of the disk driver given in a list, when you call INSTALL without parameters. The third up to the fifth assignment are optional parameters.

The <base address> is the base address of the card as jumpered in system. The fourth parameter <number of desired disks | <P>partition list> allows you to select only one or more physical disks (FLOPPY/HARD DISKS) or by preceding a P to select one or more logical disks. If no fourth parameter is given the driver will handle all disks are found (maximum 2 FLOPPY DISKS and 4 HARD DISKS for each driver). The fifth parameter <partition offset> is an offset added to all logical partition numbers for that driver. Each installable disk file must have a specific structure on top of file that helps INSTALL to handle them. There are two structures handled by VMEPROM. If there is any write protect for the object code of the disk driver (i.e. the code is in EPROM), the driver file must have the following structure:

```
WBEG    DC.W    'W0'     ; IDENTIFIER
        BRA.S   INIT     ; INIT DISK
        BRA.L   XDOF     ; DISK OFF
        NOP
        NOP
        NOP
        BRA.L   XREAD    ; READ SECTOR
        NOP
        NOP
        NOP
        BRA.L   XWRIT    ; WRITE SECTOR
        NOP
        NOP
        NOP
        DC.B    'WSAMPLE',0
        EVEN
```

If there is no write protect the driver file can also have the following structure (like as used by PDOS), and VMEPROM will overwrite all BSR with a BRA.

```
WBEG    DC.W    'W0'     ; IDENTIFIER
        BRA.S   INIT     ; INIT DISK
        BSR.L   XDOF     ; DISK OFF
        JMP     $0.L     ; OLD DISK OFF ROUTINE
*                        ; (PROVIDE ADDRESS AT INSTALL TIME)
        BSR.L   XREAD    ; READ SECTOR
        JMP     $0.L
        BSR.L   XWRIT    ; WRITE SECTOR
        JMP     $0.L
        DC.B    'WSAMPLE',0
        EVEN
```

The driver file always starts with an identifier "W0" and after the little jump table INSTALL assumes a string, null terminated, which describes the driver.

The initialization routine has the following inputs and outputs:

```
INIT  -  INSTALL DISK DRIVER

        IN: A1.L = K1$BEGN
            A2.L = OPTIONAL CARD BASE ADDRESS
            D7.W = OPTIONAL DISKNR (BY VMEPROM SET TO FFFF)
       OUT: D0.W = -1 ERROR
                   NUMBER OF CARDS
```

NOTE: The UART for the I/O devices on-board of the CPU card are installed by default, but a disk driver is only installed by default if set by the front panel switches.

Example:

```
? INSTALL ?

   THE FOLLOWING UARTS AND DISK DRIVER ARE ALREADY IN EPROM:

   UART    ONBOARD_20               ADDR: $FF005000
   UART    FORCE SIO-1/2            ADDR: $FF005400
   UART    FORCE ISIO-1/2           ADDR: $FF005800
   DISK    FORCE ISCSI-1            ADDR: $FF005C00
   DISK    FORCE WFC-1              ADDR: $FF006400

? INSTALL

THE FOLLOWING DRIVERS ARE INSTALLED:

UART   NAME        BEGINADDRESS   PROCESSOR
U1     ONBOARD_20  $FF005000      68020/30

DISK  NAME    BEGINADDRESS   F/W FIRSTDISK(W) PHYSICAL DISK

? INSTALL W,80C500,,P3/4/9-11,30
  DISK DRIVER FORCE ISCSI-1 INSTALLED

? INSTALL

THE FOLLOWING DRIVERS ARE INSTALLED:

UART   NAME              BEGINADDRESS   PROCESSOR
U1     ONBOARD_20        $FF005000      68020/30

DISK   NAME           BEGINADDRESS   F/W   FIRSTDISK(W) PHYSICAL DISK
DRV0  FORCE ISCSI-1   $FF005C00      2/1   33           F0,F1,W0-W3
```

1.2.44 KM - KILL MESSAGE

```
Format: KM
        KM <task #>
```

The KM command removes all task messages associated with <task #> from the message buffers.

If no task is specified, then all messages associated with the current task are deleted from the message buffers.

See also 1.2.65 SM - SEND MESSAGE.

1.2.45 KT - KILL TASK

```
Format: KT
        KT {-}<task #>
```

The KILL TASK command removes a task from the task list and returns the task's memory to the free pool for use by other tasks. Only your current task or a task spawned by your task can be killed. (Task 0 can kill any task except itself or a task that is kill protected.)

Each task is assigned a unique task number which is shown by the LIST TASK command. Only the current task (indicated by '*') or those spawned by the current task (indicated by current task number following a "/" character) may be killed. Task #0 is the system task and cannot be killed.

If a minus sign (-) precedes the task number, then the task's memory is not deallocated to the memory bit map. If the task number is zero, then the current task is killed without deallocating memory.

If no parameter is given, then the current task is killed and memory is deallocated.

All open files associated with the killed task are closed by the KT command.

Example:

```
? LT
task    pri   evl/ev2   size      tcb       eom       ports
name
*0/0    64              352   00007000 0005D000  1/1/0/0/0   lt
 1/0    64      98      100   0005D000 00076000  2/2/0/0/0

? KT 1
? LT
task   pri  evl/ev2  size     tcb       eom          ports
name
*0/0    64              352   00007000 0005D000  1/1/0/0/0   lt

?
```

1.2.46 LC - LIST DIRECTORY

Format: LC <file list>

The LIST DIRECTORY command displays a selected list of disk file names. The file names are printed in a compressed format with 5 names on every line.

The files are selectively listed according to file name, extension, level, disk number, file attribute, or date of last change.

The format of the <file list> is defined as follows:

```
  <file list> = {file}{:ext}{;level}{/disk}{/select...}
where:  {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
        {:ext} = 1 to 3 characters (:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255
     {/select} = /AC = Assign Console file
                 /BN = Binary file
                 /BX = VMEPROM BASIC token file
                 /EX = VMEPROM BASIC file
                 /OB = 68000 VMEPROM object file
                 /SY = System file
                 /TX = Text file
                 /DR = System I/O driver
                 /*  = Delete protected
                 /** = Delete and write protected
                 /Fdy-mon-yr = selects files with date of
                               last change greater than
                               or equal to 'dy-mon-yr'.
                               /Fmn/dy/yr format can also
                               be used.
                 /Tdy-mon-yr = selects files with date of
                               last change less than or
                               equal to 'dy-mon-yr'.
                               /Tmn/dy/yr format can also
                               be used.
```

In the file list specification, the '@' character indicates all subsequent characters match and the '*' character is a single character wild card.

Also displayed with each directory listing is the disk name, the number of files stored on the disk and the number of directory entries available. This information is useful in disk maintenance. The directory entries are not necessarily in alphabetical order but in the order they are stored in the disk directory.

See also: 1.2.49 LS - List directory sequential

Example:
```
? LC
  test          lv             ls            lc
Number of files: 4             Sectors allocated:  5
?
```

1.2.47 LD - LOAD FILE

```
Format: LD <file name>
        LD <file name>,<start address>
```

The LOAD FILE command loads a file into memory but does not begin executing it. The file must be of the type 'SY'. The starting load address is optionally specified by <start address>. Otherwise it defaults to immediately following the TCB.

This command can be used to debug files, load multiple files or to load programs outside of known tasking memory.

The LOAD FILE command uses the XLDF primitive and loads 'SY' files four bytes at a time. As a result, as many as three extra bytes may be loaded.

Example:

```
? ld test1,8000
? di 8000 5
8000    NOP
8002    NOP
8004    NOP
8006    NOP
8008    NOP

?
```

1.2.48 LO - Load S-record

Format: LO
        LO <address> , <command line>,<-V|-T>

The LO command loads a S-rec·-d into memory from the standard input port. Normal I/O redirection may be used for input from other ports. The starting load address is optionally specified by <address>.

An optional command line may be specified which is sent to the host before the load of the S - record starts. This can be used to initiate the download in the host system, without having to use the TM Command.

There are two possible options which must be proceeded by a minus sign. If option V is given, the contents of the S-records will only be compared with the contents of those memory locations which are to be loaded.

The different values of the memory locations and the S-record data are displayed.

If option T is given without an address parameter, the S-records are loaded immediately following the TCB.

The following S-record types are supported by VMEPROM:

S0	Start record, it is ignored by VMEPROM and may be omitted.
S1	Data record with 16 bit address field
S2	Data record with 24 bit address field
S3	Data record with 32 bit address field
S7	End record with 32 bit address field
S8	End record with 24 bit address field
S9	End record with 16 bit address field

If the address for the LO command is specified on the command line, the address fields in the data records are ignored and the S-record is loaded contiguously from the specified address upwards.

If the address field of the end record is equal, 0 control is transferred back to the command interpreter of VMEPROM. If the address file holds an address, VMEPROM automatically executes a "G address" command after the S-record is loaded and an end record is found. Because of the "G" command all breakpoints which are defined are inserted in the program.

See also: 1.2.26 DU - Dump S-records

Example:

```
? lo <2 8800
?
```

1.2.49 LS - LIST DIRECTORY

Format: LS <file list>

The LIST DIRECTORY command displays a selected list of disk file names. The file listing also includes the directory level, file type, file size, start sector address, date of creation, and date of last update.

The files are selectively listed according to file name, extension, level, disk number, file attribute, or date of last change.

The format of the <file list> is defined as follows:

```
<file list> = {file}{:ext}{;level}{/disk}{/select...}
where:  {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
        {:ext} = 1 to 3 characters (:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255
     {/select} = /AC = Assign Console file
                 /BN = Binary file
                 /BX = VMEPROM BASIC token file
                 /EX = VMEPROM BASIC file
                 /OB = 68000 VMEPROM object file
                 /SY = System file
                 /TX = Text file
                 /DR = System I/O driver
                 /*  = Delete protected
                 /** = Delete and write protected
                 /Fdy-mon-yr = selects files with date of
                               last change greater than
                               or equal to 'dy-mon-yr'.
                 /Fmn/dy/yr format can also be used.
                 /Tdy-mon-yr = selects files with date of
                               last change less than or
                               equal to 'dy-mon-yr'.
                 /Tmn/dy/yr format can also be used.
```

In the file list specification, the '@' character indicates all subsequent characters match and the '*' character is a single character wild card.

Also displayed with each directory listing is the disk name, the number of files stored on the disk and the number of directory entries available.

This information is useful in disk maintenance.

The directory entries are not necessarily in alphabetical order but in the order they are stored in the disk directory.

See also: 1.2.46 LC - List Directory

Example:

```
? LS
Lev  Name:ext    Type      Size      Sect    Date created     Last update
102  test          C         1     013B  00:50 16-Mar-88 00:51 16-Mar-88
102  lv           +C         1     0145  00:56 16-Mar-88 00:56 16-Mar-88
102  ls            C         1     0146  00:56 16-Mar-88 00:56 16-Mar-88
Number of files: 3            Sectors allocated:  3

?
```

1.2.50 LT - LIST TASKS

Format: LT

The LT command displays all tasks currently in the task list to the console. Task 0 is the system task and is created automatically during system initialization. This task cannot be killed.

Your current task is indicated by an '*' preceding the task number. Following the task number is a slash and the parent task number. Subsequent data provides the current status of each task and is defined as follows:

task	{*=current}Task #/parent task #
pri	Task priority (1-255)
ev1/ev2	Suspended event(s)
size	Task size (k bytes)
tcb	Task control Block
eom	End of memory
ports	Task I/O ports in the following order: input port/output port/Unit 2 port/Unit 4 port/Unit 8 port
name	The name of the command currently executing

Example:

```
? LT
task   pri  ev1/ev2  size     tcb       eom       ports      name
*0/0   64            352    00007000 0005D000   1/1/0/0/0  lt

?
```

1.2.51 <u>LV - DIRECTORY LEVEL</u>

```
Format: LV
        LV <level>
```

The DIRECTORY LEVEL command displays or sets the current directory level used in directory listings and file definitions.

The DIRECTORY LEVEL command without any argument displays the current directory level. A file defined without a specified directory level is defined on the current level.

If an argument is specified, it is converted to a number and sets the current directory level.

The range is from 0 to 255 in decimal.

The disk directory is soft partitioned into 256 different groups, facilitating file maintenance. A soft partition means that any file is accessible from any current level. It also means that file names must be unique for each disk number (disk directory).

Example:

```
? LV
Level = 103

? LV 100
? LV
Level = 100

?
```

1.2.52 <u>M - Modify Memory</u>

```
Format:  M <address>[,<option>]
         MM <address>[,<option>]
```

Option is B | W | L & N & O | E

The Modify Memory command is used to inspect and change memory locations. Several options are allowed on the command line to specify the size of the memory and the access type. The following options are allowed:

B	memory is byte sized (8 bits).
W	memory is word sized (16 bits). This is the default.
L	memory is long word sized (32 bits).
O	memory is byte sized and on odd addresses only.
E	memory is byte sized and on even addresses only.
N	memory is write only, the current contents is not displayed.

The Odd and Even options are overriding the B/W/L options. The N (no read) option has to be specified after the size qualifier and after the Odd/Even specification. All memory accesses check that the write access was successful by performing a read after the write unless N is specified. If the data written and the data read do not match, the command is terminated and an error message is displayed.

The memory modify command supports a number of sub-commands, which can be entered instead of a new memory value. These sub-commands do not change the access option specified on the command line.

The following sub-commands are supported:

<cr>	open next location
=	open same location again
-	open previous location
-<count>	go back <count> bytes
+	open next location
+<count>	go forward <count> bytes
#<address>	open new absolute address
?<mnemonic>	insert 68000 opcode at current address
.	exit to the command interpreter

Example:

```
? M 8000
8000   4246 : <cr>
8002   1C2E : <cr>
8004   0441 : <cr>
8006   4247 : ?nop<cr>
8008   A05A : -2<cr>
8006   4E71 : -<cr>
8004   0441 : #8000<cr>
8000   4246 : <cr>
8002   1C2E : .
?
```

1.2.53 MD - Display Memory

Format: MD <address>
        MD <address>[,<count>]

The MD command displays the memory contents of the specified address. The data is displayed in hex and ASCII representation, 16 bytes on every line. If the hex value cannot be displayed in ASCII representation, a full stop (".") is displayed instead.

If no count is specified on the command line, the Display Memory command displays 16 lines, representing 256 bytes of data, and prompts the user to display more or to return to the command interpreter.

If a carriage return (<cr>) is entered, the next 256 bytes are displayed. Any other character returns control back to the command interpreter of VMEPROM.

If a count is specified on the command line, the value is interpreted as the number of bytes to be displayed. All values are assumed to be in hex.

Example:

```
? MD 8000 30
8000  42 46 1C 2E 04 41 4E 71  A0 5A 63 12 A0 56 63 08   BF...ANq.Zc..Vc.
8010  3C 01 A0 5A 63 34 60 2A  A0 8C 02 5B A0 0E A0 8C   <..Zc4'*...[....
8020  01 C0 A0 8C 01 F1 A0 80  66 0A 42 81 32 06 A0 50   ........f.B.2..P

?
```

1.2.54 MF - MAKE FILE

Format: MF <file>

The MF command allows an ASCII file to be created from the user console. The <file> must be previously defined or preceded by a '#'. The normal line editing is permitted but once a return key has been entered, the line is written to the file.

A [CTRL-C] cancels the line without writing it to the file. An [ESC] terminates the process, closes the file, and returns to the VMEPROM monitor.

The MF command uses the XGLB primitive and hence, normal editing control characters are available and lines are limited to 78 characters. Control characters other than those used for editing cannot be entered (i.e. this includes a TAB character.)

Example:

```
? MF test
This is a test file to show the<cr>
functionality<cr>
of<cr>
the MF command.<cr>
<esc>

? SF test
This is a test file to show the
functionality
of
the MF command.

?
```

1.2.55 MS - Set Memory to Constant or String

Format: MS <address>,<data|"string">

This command writes the specified data pattern to memory. The data may consist of hex numbers and ASCII data in any combination. The ASCII data must be put in inverted commas.

Example:

```
? bf 8000 8100 ff b
? ms 8000 "Hello World"0d0a00
? md 8000 20
8000  48 65 6C 6C 6F 20 57 6F  72 6C 64 0D 0A 00 FF FF   Hello World.....
8010  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF   ................
?
```

1.2.56 PROMPT - CHANGE PROMPT SIGN

Format: PROMPT [<data|"string">]

The PROMPT command is used to change the prompt for the current task in the used specified pattern.

The data may consist of hex numbers and ASCII data in any combination. The ASCII data must be put in inverted commas.

If no parameter is given, the default VMEPROM prompt "?" will occur. The user defined prompt sign will be truncated to nine characters maximum.

Example:

```
? PROMPT "#"_
#_

#PROMPT ("HELLO> ")_
HELLO> _

HELLO> PROMPT_
? _
```

1.2.57 <u>RC - RESET CONSOLE</u>

Format: RC

The RESET CONSOLE command is used in an Assigned Console (type=AC) file to terminate the procedure and to revert back to the system console. This allows for a graceful termination of the file commands by closing the file and prompting for a new command.

Since procedure files can be nested, only the current procedure file is closed.

1.2.58 <u>RD - RAM DISK</u>

```
Format:  RD
         RD {-}<unit>[,<size>][,<address>]
```

The RAM DISK command sets or displays the current RAM disk's units, sizes and memory addresses. VMEPROM maintains a RAM disk list, providing up to 4 RAM disks at any time. Each RAM disk has a unique disk number and separate memory address. The RAM disk command allows you to add RAM disks, delete, renumber and list them. When the address or the size is changed, the RAM disk must again be initialized. This is easily be done by preceding the RAM disk unit by a minus sign. Otherwise, the INIT command can be used to initialize the disk.

The default Ram disk setup of VMEPROM is described in the User's Manual of your CPU - board. If there is no parameter the current RAM disks are listed showing disk number, size number in sectors and base address. They may not appear in the order defined.

The first assignment <unit> specifies the disk number to be used for the RAM disk. It must be in the range of 0-99.

The argument <size> specifies the size of the RAM disk in sectors. Each sector has a size of 256 bytes. The given size will be rounded up to 2 Kbyte boundary. So a RAM disk of 32 Kbytes will have a size of 128 sectors. If the second parameter <size> equals zero, then the RAM disk <unit> is removed from the list. To aid with memory management, if the <unit> was positive or zero, then the memory that was used by that RAM disk is deallocated in free memory pool for new tasks or other RAM disks. If <unit> was negative, the memory is not deallocated. If the second parameter <size> is non zero, then either a new RAM disk is to be entered into the list or an existing RAM disk is to be renumbered.

If there is no third assignment <address>, then a new RAM disk is created of <size> sectors coming from either the free memory pool, if possible,or from the calling task's memory. If there is a third parameter <address>, then VMEPROM tries to find <address> among the currently defined RAM disks. If there is a match, the new <unit> and <size> replace those of the current disk at <address>. (no check is made that <size> is the same.) If there is no matching address, then the new RAM disk is entered in the list, but no memory management is performed.

Example:

```
? RD
Ram disk unit = 8, size = 128, address = $00077DFC

? RD -50,100,$800000
? RD
Ram disk unit = 8,  size = 128, address = $00077FDC
Ram disk unit = 50, size = 104, address = $00800000
```

1.2.59 RM - Modify Processor Registers

```
Format:  RM
         RM <register>
         RM <register>,<value>
```

The RM command modifies the processor registers or, if available, the data registers of the 68881 coprocessor. Three modes are allowed.

The first mode is an interactive mode, which scans all registers. For each register, the current value is displayed and the user is prompted to enter a new value. A <cr> leaves the register unchanged. After a new value or a <cr> has been entered, the same procedure will be started for the next register. If an <ESCAPE> or <.> has been entered, control is transferred back to the command interpreter.

The second mode needs only requires a change in the register specified. The current value is then displayed and the user is prompted to enter a new value. A <cr> leaves the register unchanged. After a new value or a <cr> has been entered, control is transferred back to the command interpreter.

The third mode allows the specification of the new new value for the given register on the command line and does not display the the old value.

The following registers may be modified by the user:

VBR	Vector base register, only on 68010/68020/68030 systems
SFC/DFC	Source and Destination function code register
CAAR	CACHE address register, only for 68020/68030 systems
CACR	CACHE control register, only for 68020/68030 systems
PC	Program counter
SR	Status register
USP	User Stack pointer
SSP	System Stack pointer
MSP	Master Stack pointer, only on 68020/68030 systems
D0-D7	Data registers D0-D7
A0-A7	Address registers A0-A7, where A7 is the current stack pointer as defined by the status register
FP0-FP7	Floating point Coprocessor registers, if available.

Caution: Be careful when modifying the Vector Base register (VBR) as VMEPROM is a interrupt driven system and any modifications to this register may crash the system.

Example:

```
? RM D0
D0 = 00000000  : 12345678<cr>

? RM D1 1000
? DR
      0        1        2        3        4        5        6        7
D: 12345678 00001000 00000000 00000100 000066DC 00000010 00000006 00000000
A: 00006290 0005D6FC 00005000 00006297 0005D6F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 .....  SFC  = 0  DFC = 0

? RM FP0
FP0 =  0.00000000 E+000  : 1234.56E-24<cr>

? DRF
FP0: 1.23456000 E-021 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000
FP4: 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000
?
```

1.2.60 RN - RENAME FILE

```
Format: RN <file1>,<file2>
        RN <file1>,<level>
```

The RENAME FILE command changes the file name stored in the disk file directory. The RENAME command may also be used to move a file from one directory level to another. The file <file1> is renamed to <file2>. A disk specification in the second parameter is meaningless. If a number <level> is used instead of <file2>, the <file1> is moved to the new level.

Example:

```
? lc
  temp          rn1
Number of files: 2            Sectors allocated:  2

? rn temp,temp1
? lc
  temp1         rn1
Number of files: 2            Sectors allocated:  2

?
```

1.2.61 RR2 - EPROM Programming

```
Format: RR2 [<f>,<file>],<board>,<mode>,<option>
        RR2 [<m>,<addr>,<cnt>],<board>,<m>,<opt>
```

The RR2 command is used for programming EPROMS or EPROMS on a SYS68K/RR-2/RR-3 board. It can also be used to transfer files or memory contents into a SRAM area on the RR_2 or to load EPROM/EEPROM contents into the VMEPROM memory.

The following are examples on the usage of the RR2 command:

```
   ? RR2 F,FILENAME,RR_2_ADDRESS,MODE,OPTION
     if the source is a disk file, or

   ? RR2 M,STRTADDR,BYTECNT,RR_2_ADDRESS,MODE,OPTION
     if the source is in memory.
```

The following describes the parameters:

```
F,FILENAME.............source = disk file
                            F = source flag
                     FILENAME = the name of the source file
M,STRTADDR,BYTECNT.....source = memory
                            M = source flag
                     STRTADDR = source start address
                      BYTECNT = source length in bytes
RR_2_ADDR..............the address of the RR_2 bank
MODE...................1 = 8 bit mode (single EPROM)
                       2 = 16 bit mode (two EPROMS)
                       4 = 32 bit mode (four EPROMS)
OPTION.................P = program an EPROM (includes E and V
                                 and a bit test)
                       E = check if EPROM is empty
                       V = verify source and EPROM contents
                       L = load EPROM contents to memory
```

For further information on the hardware setup of the SYS68K/RR2 or SYS68K/RR3 board please refer to the user's manual of the RR-2/3 board.

Example:

```
  ? RR2 M,$0,$8000,$800000,2,E
```

executes an empty check in word mode for EPROM type 27128 (16k x 8) at RR_2 address $800000. The M - source flag and the memory address are dummy.

? RR2 F,PROG/2,$800000,4,P
programs EPROMS at address $800000 in 32-bit mode with the source file PROG from disk 2.

? RR2 M,$10000,$2000,$800000,1,L loads the contents of an 8k x 8 EPROM at address $800000 into the memory to address $10000.

SYS68K/RR-2/RR-3 board configuration:

This example contains the RR-2 board configuration and and the program usage for 27128 EPROMs in the 16 bit mode.

<u>Jumper settings for 16k x 8 EPROMs on bank 2 (TOSHIBA 27128):</u>

```
B1b        Read time selection on bank 2

           8     5
           o o o o
             I I                      250 ns
           o o o o
           1     4

B2b        Write time selection on bank 2

           3       15
           o o o o o
               I
           o o o o o                   50 ms

           o o o o o
           1       13

B4b        Device type bank 2

           4
           o o
             I                        EPROM type 1
           o o
           1

B13b       Device size bank 2

           10      6
           o o o o o
           I I I I                    4 x 16k x 8
           o o o o o
           1       5


B15        Device pinning bank 2

           3                   33
           o o o o o o o o o o o
           I       I
           o o o o o o o o o o o
               I           I I
           o o o o o o o o o o o
           1                   31
```

Page 88 was blank

```
B16        Enable VPP generator

           2
           o
           I
           o
           1

B17        Select VPP bank 2

           3
           o
           I
           o                                21V

           o
           1

B18        Select output enable on VPP bank 2

           2
           o

           o
           1

B19        Select chip erase bank 2

           3
           o

           o
           I
           o
           1

B11        Upper address bank 2

           2     8
           o o o o
             I I I                          $8
           o o o o
           1     7

B12        Lower address bank 2

           2     8
           o o o o
           I I I I                          $0
           o o o o
           1     7
```

Program call for subsequent jobs:

a) EPROM empty check

```
   ? RR2 M,$0,$8000,$800000,2,E
         \ \  \       \    \ \__option = empty check
          \ \  \       \    \___mode = word
           \ \  \       \_______RR-2 base address
            \ \  \______________byte count (2 EPROMs 16k x 8)
             \ \________________memory address (don't care)
              \_________________source = memory
```

b) program EPROMs

```
   ? RR2 F,MYFILE:PRG/4,$800000,2,P
         \      \         \    \ \__option = program
          \      \         \    \___mode = word
           \      \         \_______RR-2 base address
            \      \________________source file name
             \______________________source = file
```

c) load EPROMs into memory

```
   ? RR2 M,$10000,$8000,$800000,2,L
         \   \     \      \    \ \__option = load
          \   \     \      \    \___mode = word
           \   \     \      \_______RR-2 base address
            \   \     \_____________byte count(2 EPROMs 16k x 8
             \   \__________________memory address
              \_____________________destination = memory
```

1.2.62 RS - RESET DISK

Format: RS
        RS <disk #>

Disk files must be closed at the end of any task so that sector buffers are flushed to the disk, pointers updated in disk directories, and file slots released for further usage. The RESET command either closes all open files associated with your task or closes all open files on a specified disk. The first mode allows your task to terminate itself without affecting the files of other tasks, while the second mode is used before withdrawing a disk from a disk drive.

RESET also clears the assigned console FILE ID (ACI$(A6)).

However, the assigned console file may not be closed if the RESET disk option is used and the file resides on a different disk.

Example:

```
? FS
Slot  Name        ST   SM   PT       SI   EOF     TN   BF       FLGS
64    fs1;101/6   C104 0142 00003916 0000 0000/82 0000 0000389E 00000000
? RS
? FS
Slot  Name        ST   SM   PT       SI   EOF     TN   BF       FLGS

?
```

1.2.63 SA - SET FILE ATTRIBUTES

```
Format: SA <file>
        SA <file>,<attributes>
```

The SET FILE ATTRIBUTES command associates file attributes with a file in the disk directory. File attributes include file types and protection flags.

The following file types are supported by VMEPROM:

AC	Assign Console, command file.
SY	680x0 executable file, memory image.
TX	Text file.
DR	Loadable driver.
C	Contiguous file. This type can not be set or reset by the user.
*	Delete protected. Allowed in addition to other types.
**	Delete and write protected. Allowed in addition to other types.

Note: The file type "C" is an addition to the other file types and is set by VMEPROM. It cannot be set or cleared by the user.

The following types are not decoded or used by VMEPROM but may be used:

BN	Binary file.
EX	Basic file.
BX	Basic file.
OB	VMEPROM object file.

The following gives a detailed description of all file types supported by VMEPROM:

1. AC - Assign console. A file typed 'AC' specifies to VMEPROM that all subsequent requests for a console character will be intercepted and the character obtained from the assigned file.

2. SY - System file. A 'SY' file is generated automatically by the Save File to Memory command. Also the LD (Load file to memory) command checks for the SY type. If any program shall be loaded from disk to memory and be executed, it must have the type SY.

3. TX - ASCII text file. A 'TX' type classifies a file as containing ASCII character text.

4. DR - System I/O driver. A 'DR' file type is a VMEPROM system I/O driver. Channel buffer data is treated as a program and is used to extend the file system to I/O devices. The Loadable I/O drivers are described in detail in the Appendix.

5. * - Delete protect. The file is delete protected and cannot be deleted from the disk. This file type is an addition to the other file types.

6. ** - Delete and write protect. The file cannot be deleted or written to by any system call. This file type is an addition to the other file types.

Example:

```
? SA FILE         Clears all attributes (except 'C')
? SA FILE,SY      Sets SY type only
? SA FILE,**      Sets protection only
? SA FILE,OB**    Sets type and protection

 ? LS
   Lev  Name:ext Type  Size  Sect  Date created     Last update
   101  templ     +C      1  0110  19:47 16-Mar-88  19:47 16-Mar-88

Number of files: 1             Sectors allocated:  1

? SA templ TX
? LS
Lev  Name:ext  Type   Size   Sect  Date created     Last update
101  templ     TX+C     1    0110  19:47 16-Mar-88  19:47 16-Mar-88

Number of files: 1             Sectors allocated:  1

?
```

1.2.64 SF - SHOW FILE

Format: SF {-}<file name>

The SHOW FILE command displays the disk file as specified by <file name> on your console. The output is paged and truncated to 78 characters per line unless the file name is preceded with a minus sign. Pressing [ESC] terminates the command at any time.

If a minus sign precedes the file name, then the file is displayed without line truncations or paging. Again, [ESC] terminates the command.

Example:

```
? SF TEST
This is a test file to show the
functionality
of
the MF command.

?
```

1.2.65 SM - SEND MESSAGE

Format:
```
        SM <task #>,<message>
```

The SEND MESSAGE command puts an ASCII text message in a message buffer. The destination is specified by <task#>. The message can be up to 63 characters in length.

If a message is sent to itself, i.e. the task which is sending the message, the complete message is interrupted as a command line and executed.

Note: No other commands can be appended to an 'SM' command with a period, since the <message> parameter takes everything up to the carriage return.

See also: 1.2.44 KM - KILL MESSAGE.

Example:

```
? SM 2,Hallo
? SM 0, LV
? LV
Level=1
?
```

1.2.66 <u>SP - DISK SPACE</u>

```
Format: SP
        SP <disk #>
```

The DISK SPACE command displays the current number of defined files, the total possible directory size, the number of disk sectors free, the largest possible contiguous file, the number of disk sectors used, and the number of allocated disk sectors.
All numbers represent decimal sectors. The total size in bytes is the number of sectors times 252.

The <disk #> specifies the disk number. If no parameter is used, then the default disk is displayed.

The 'Files' parameter lists the current number of defined files in the disk directory. This is followed by the maximum number of files definable in the directory.

The 'Free' parameter shows the number of sectors still available for file storage. This is followed by the largest number of contiguous sectors. This is helpful in defining contiguous files.

The 'Used' parameter shows exactly how much of the disk is truly used versus the amount of disk storage allocated. Some files may have END-OF-FILE markers pointing within the file and not at the end. If these files were copied to another disk, the unused storage would be recovered.

Example:

```
? SP 6
Files= 16/128
Free = 2080/1596
Used = 288/293

?
```

1.2.67 <u>ST - SET TASK TERMINAL TYPE</u>

```
Format: ST
        ST <type>
```

The ST command sets the position cursor (PSC$) and clear screen (CSC$) variables in the task control block (TCB). This command makes it easy to use various types of terminals together with VMEPROM. Each task has its own characters for these two functions, which are initialized, when the task is started, to the parent task control set.

If a legal <type> is passed in the command line, then ST simply enters the corresponding sequences into the user status block.

Otherwise, the command prints the following table of options:

```
      D = VT52
      L = Lear Siegler ADM3a
     ✓V = VT100
      T = TVI 950
      U = User defined
   Type = _
```

and prompts the user for an input. Enter the letter representing the type of terminal you are using.

The terminal type setup is only required for the VMEPROM screen editor. No other function uses the terminal dependant sequences.

The default setup of VMEPROM is the codes for a VT52 terminal.

In addition to the built in terminal types, the ST command allows to enter the values for position cursor, clear screen, clear to end of screen and clear to end of line interactively with the "C" option. So nearly every terminal can be used with VMEPROM.

```
? St U  to to enter a user defined terminal

Enter encoded position cursor value: $.
```

Now the position cursor code can be entered in hex. The hex value must be 16 bit. The format of the leading characters for cursor positioning is as follows (note that each letter represents a bit):

```
B111 1111 0222 2222

B = 0 then $00 bias
    1 then $20 bias
O = 0 then row before column, 1 then column before row
1 = 7 bits for first ASCII lead in character
2 = 7 bits for second ASCII lead in character
```

A value of 0 will result in the code for a VT100 terminal.

Enter encoded clear screen value: $_

The cursor home and clear screen can also be entered as a encoded 16 bit value. The format is (note that each letter represents a bit):

```
  E111 1111 E222 2222

  E = if 1 then precede with [ESC]
  1 = 7 bits for first ASCII character
  2 = 7 bits for second ASCII character
  If all 16 bits are 0 then a VT100 is selected
```

Enter encoded clear to end of screen value: $.

This is the code to clear the access from the current cursor position to end of screen. The format is:

```
  0111 1111 0222 2222

  1 = 7 bit for first ASCII character
  2 = 7 bit for second ASCII character
```

Enter encoded clear to end of line value: $_

This is the code to clear from the cursor position to the end of the line. The format is:

```
0111 1111 0222 2222

1 = 7 bit for first ASCII character
2 = 7 bit for second ASCII character
```

Example:

```
? ST
   D = VT52
   L = Lear Siegler ADM3a
   V = VT100
   T = TVI 950
   U = User defined
Type = L

? ST D
?
```

1.2.68 SV - Save Memory to File

Format: SV <begin>,<end>,<filename>

The SAVE TO FILE command writes binary memory images to the file specified by <file>. The parameters <begin> and <end> specify the start and end memory bounds.

The file is created on the disk if it does not exist. The file gets the file type 'SY'.

Example:

```
? SV 8000 8100 file
?
```

1.2.69 <u>SY - SYSTEM DISK</u>

```
Format: SY
        SY <disk1>{,<disk2>{,<disk3>{,<disk4>}}}
```

The disk device identifier is contained within the file name.

However, a default or system disks are assigned by the SY command.
On all open and define commands, file names without the disk identifier follow the system disk specification order in looking for the file on disk. All other commands use only the first system disk specification.

Example:

```
? SY
System disks : 6

? SY 6,2
? SY
System disks : 6,2

?
```

1.2.70 <u>T - Trace Program Execution</u>

```
Format:   T
          T <address>
          TT
          TT <address>
```

The TRACE command starts a user program in trace mode. The start address may be specified in the command line. If omitted, the current value of PC as displayed by the DR command is used. The number of instructions to be traced are defined by the TC (set trace count) command. The default after reset is 1 instruction.

After every instruction, the contents of the processor registers is displayed along with the disassembled code of the instruction executed. If no Trace Count is set or it reached 0, the user is prompted to continue the trace or return to VMEPROM. Tracing can be continued by entering a space (" ") or a carriage return (<cr>).

See also: 1.2.71 TC Set Trace Count
          1.2.73 TJ - Trace on change of flow

Example:

```
? T 8000
Trace
       0        1        2        3        4        5        6        7
D: 12345678 00000000 00000000 00000100 000066DC 00000010 00000006 00000000
A: 00006290 0005D6FC 00005000 00006297 0005D6F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0
8002 : MOVE.L #$1234,D1<cr>
Trace
       0        1        2        3        4        5        6        7
D: 12345678 00001234 00000000 00000100 000066DC 00000010 00000006 00000000
A: 00006290 0005D6FC 00005000 00006297 0005D6F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0
8008 : MOVE.L D1,D7<cr>
Trace
       0        1        2        3        4        5        6        7
D: 12345678 00001234 00000000 00000100 000066DC 00000010 00000006 00001234
A: 00006290 0005D6FC 00005000 00006297 0005D6F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0
800A : LEA.L ($8100,PC),A0<cr>
Trace
       0        1        2        3        4        5        6        7
D: 12345678 00001234 00000000 00000100 000066DC 00000010 00000006 00001234
A: 00006100 0005D6FC 00005000 00006297 0005D6F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0
800E : MOVE.L (A0),D0<cr>
```

```
Trace

        0        1        2        3        4        5        6        7
D: 4E714E71 00001234 00000000 00000100 000066DC 00000010 00000006 00001234
A: 00006100 0005D6FC 00005000 00006297 0005D6F0 000066DC 00007000 0005D7FC
VBR  = 00000000  CAAR = 00000000   CACR = 00000000 . . . .
*USP = 000767FC  SSP  = 00007BB2   MSP  = 00007890
PC   = 00008000  SR   = 0000 ..... SFC  = 0  DFC = 0
8010 : NOP<esc>

?
```

1.2.71 <u>TC - Set Trace Count</u>

Format: TC <count>

The Set Trace Count command sets the number of instructions to be traced continuously. The default after reset is 1.

See also: 1.2.70 T - Trace program execution
          1.2.73 TJ - Trace on change of flow

Example:

```
? TC
Trace count = 0

? TC 100
? TC
Trace count = 100

?
```

1.2.72 TIME - Enable/Disable Display of the Program Run Time

```
Format: TIME
        TIME ON
        TIME OFF
```

VMEPROM has the ability to measure the run time of user programs or command execution of the built in commands. This feature can be turned on and off with the TIME command. If only TIME is entered, the current status is displayed (i.e. On or OFF). VMEPROM displays the time in minutes, seconds, and tens and hundreds of seconds. If time measurement is enabled, a time stamp is taken whenever the command interpreter gets a complete input line. The timing stops when the function is executed and control is transferred back to the command interpreter.

Example:

```
? TIME
Time is off

? TIME ON
? BENCH 1 8000
Bench  1: Decrement long word in memory, 10.000.000 times
Benchmark time = 0:07.23
Programm execution time is 0:07.27

? TIME OFF
?
```

1.2.73 TJ - Trace on Change of Flow

```
Format: TJ
        TJ <address>
```

This command is only supported on 68020 versions. It traces a user program (like the Trace command), but only on instructions where a change of program flow occurs. Such instructions are for example: BRA, BSR, JMP, JSR, RTS etc.

See the Trace command for a complete description of program tracing.

See also: 1.2.70 T - Trace program execution

Note: This command is only available for 32 bit processors.

1.2.74 TM - TRANSPARENT MODE

```
Format: TM <port #>
        TM <port #>,<break>
```

The TRANSPARENT MODE command directs your current input to <port #>. Input received from <port #> is directed to your output. This command effectively allows you to access other systems as if you were a terminal.

This process continues until an [ESC] character is entered. This can be changed to another character by adding the <break> parameter.

Caution: Typing ^C twice will abort every command currently in the state of execution. This is independent of the brake character.

1.2.75 TP - TASK PRIORITY

```
Format: TP
        TP <task #>
        TP <task #>,<priority>
```

The TASK PRIORITY command allows you to change task priority of different tasks. The task number is specified by <task #> and defaults to the current task if omitted. If no priority is given the tasks current priority is displayed. Otherwise the tasks priority is changed to the given value.

The range of <priority> is 1 to 255, the latter being the highest priority. The highest priority, ready task always executes.

Example:

```
? LT
task   pri  evl/ev2  size     tcb       eom       ports       name
*0/0    64            354  00007000 0005D800  1/1/0/0/0     lt
 1/0    64     98     100  0005D800 00076800  2/2/0/0/0

? TP 0,100
? LT
task   pri  evl/ev2  size     tcb       eom       ports       name
*0/0   100            354  00007000 0005D800  1/1/0/0/0     lt
 1/0    64     98     100  0005D800 00076800  2/2/0/0/0

?
```

1.2.76 <u>UN - CONSOLE UNIT</u>

```
Format: UN
        UN <unit #>
```

The CONSOLE UNIT command sets the console output unit number. The unit number selects where the ASCII output is to be directed. Unit 1 is the system terminal. Unit 2 and 3 are auxiliary output ports. The Unit 4 is used by VMEPROM for output redirection and shall not be used.

Example:

```
? UN
Unit mask = 1

? UN 3
? UN
Unit mask = 3

? UN 1
?
```

1.2.77 <u>ZM - ZERO MEMORY</u>

```
Format: ZM
```

The ZERO MEMORY command clears the entire user work space to zeros. All flags and pointers are reset.

The memory is cleared from the end of the TCB up to the current user stack pointer. The values on the stack are not destroyed.

Example:

```
? ZM
?
```