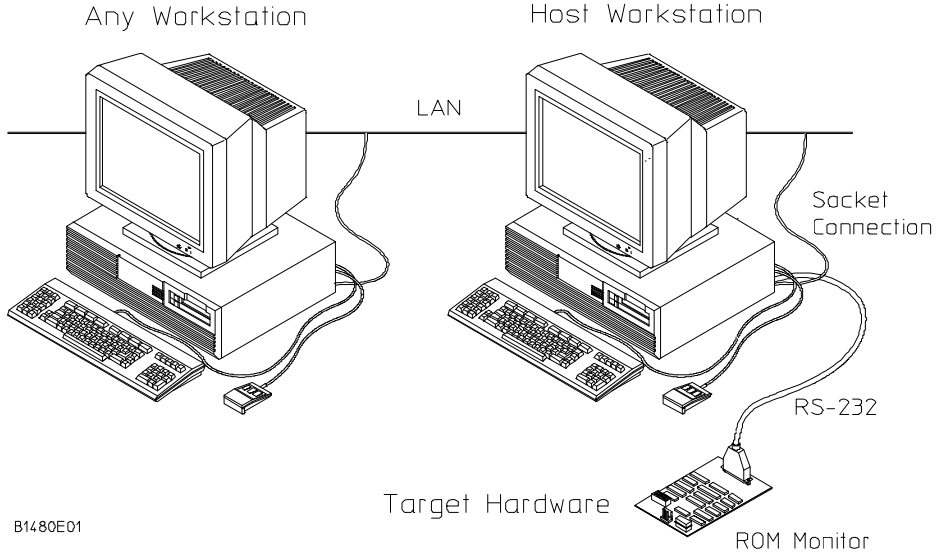
Operating Notice

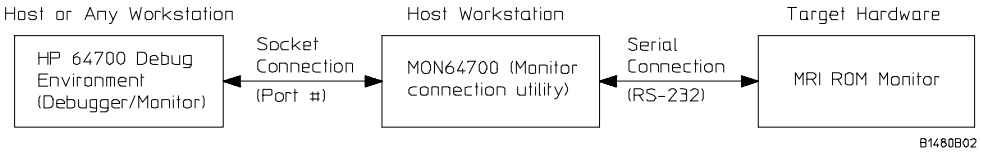
Using the MON64700 Monitor Connection Utility

Contents

Introduction

Mon64700 provides a connection between the HP64700 Debug Environment and a MRI monitor running on your target or microprocessor development board. With mon64700, you can run the HP64700 Debug Environment (Debugger/Monitor Graphical interface) on your host or on any other host where the Debug Environment software has been installed.





Mon64700 allows your workstation to simulate (act in place of) an HP 64700 emulator. When the debugger graphical interface is connected to mon64700, it works just as if it were connected to an emulator or a simulator. Mon64700 accepts the emulator commands generated by the debugger interface and translates them into the equivalent ROM monitor commands.

Mon64700 runs on a workstation which is connected to your target system via a serial cable. This is called the *host* workstation. You may use the Debugger/Monitor interface on the host workstation or on another workstation.

The Debugger/Monitor interface communicates with mon64700 by means of a *socket* connection. Sockets are communication endpoints that allow processes to communicate either locally or remotely. Sockets are accessed by means of operating system procedure calls. You do not need to understand sockets to use mon64700—these instructions will explain how to set up the socket connection.

ROM monitor limitations

Features which are not available are either "grayed out" on the menus or generate appropriate error messages. The unavailable features are:

- Simulated I/O (simio) is not supported at this time.
- Trace—ROM monitors do not provide a trace mechanism.
- Breakpoints—ROM monitors do not support breakpoints on memory reads or writes. In addition, the only 16 instruction breakpoints are allowed.
- Floating point register display—this is not supported by the MRI ROM monitor. Floating point operations, however, *are* supported.
- MMU register display—not supported by the MRI ROM monitor.
- SIM and emulator copy of SIM register comparison.

Installation

The MON64700 files are installed with the debugger software.

Setting up mon64700

Follow these steps to get mon64700 and the Debugger/Monitor interface to work with your target system:

1 Install the software.

2 Build a monitor.

3 Verify the contents of the support file.

4 Set up serial device files.

5 Verify the low-level connection to the target system.

6 Set up the host workstation files.

7 Set up additional workstations.

To install the MON64700 software

- If you have not yet installed the debugger, install it now.

  The MON64700 fileset is installed when you install the debugger software.

See also

See the debugger *User's Guide* and the *Software Installation Guide* (for Sun software) for information on how to install the software.

To build a monitor

You may need to build a custom monitor with **mct68k**. Skip this step if you already have an MRI monitor running on your target system.

If you will be using the MRI language tools:

If you will be using the MRI language tools to build the monitor, refer to the MRI manuals.

If you will be using the HP AxLS language tools:

1. Run **mct68k** to specify the parameters and create the source files for the monitor:

   ```
   mct68k -s
   ```

2. Copy the build script to your working directory:

   ```
   cp /usr/mri/mon68k/buildmon .
   ```

3. Change the permissions on buildmon:

   ```
   chmod 777 buildmon
   ```

4. Edit the buildmon file. Change all instances of the MRI language tools to HP language tools. For example, change asm68k and lnk68k to as68k and ld68k.

5. Start a C-shell:

   ```
   csh
   ```

6. Build the monitor:

   ```
   buildmon monitor
   ```

   The name of the monitor will be monitor.x, an S-Record file.

7 Exit the C-shell:

```
exit
```

You can overlay the new monitor in RAM or you may burn it into the target system ROMs.

If at all possible, you should build a MRI monitor that can respond to interrupts and that can respond to a break command coming over the serial connection. This will make the Debug Environment more responsive and eliminates the need to press the target abort button or to restart the entire system when it doesn't respond as expected or ends up in an infinite loop.

Keep the support file which you generated (the **.sup** file). In addition, if your MRI monitor overlays an existing monitor, keep this MRI monitor executable.

If you use a PC to generate the MRI monitor, you will need to transfer the resulting files to your workstation.

See also MRI MCT68K Monitor Configuration Tool documentation.

To verify the contents of the support file

- Check that your support file contains only one serial communication setup line. Here is an example of such a line:

```
S:/dev/plt_rs232_a,9600,8,n
```

- Verify that the serial port name and the baud information are correct.

  On HP 9000 Series 300 and Sun SPARCstation workstations, you may need to create a serial device to communicate with the serial port. Be sure to correctly specify the serial port name in the support file.

- Check that your support file contains a line specifying the processor type. For example, if the processor is an M68332, the support file should contain the following line:

  ```
  T:m68332
  ```

- Check that the name of the monitor executable file is a full path name. For example:

  ```
  M:$HOME/mondir/monitor.x
  ```

Support file options supported by mon64700

Option	Meaning
A	Start address
B	Buffer clearing command
D	Download command for the monitor file
M	Monitor executable file name
L	Download the monitor executable when required. Mon64700 will determine if the monitor needs to be downloaded; if the monitor is already running in the target system, it will not be downloaded.
G	Go command for auxiliary monitor
S	Serial communication setup
T	Processor type. This option has been added so that mon64700 can distinguish between the various processors which may be supported by a single debugger.
W	Character/line wait
X	Display setup information

See also

See your MRI Debugger Monitor manual for a full explanation of these options. Note that the L option behaves differently for mon67000, and that the T option has been added by HP.

To set up serial device files

If serial device files do not already exist on your host, you need to create them. Once they exist, you need to ensure that they have the appropriate permissions so that you can access them.

1 Log in as root.

2 Use the following command to find out what devices are available on the host workstation:

```
/etc/dmesg
```

3 Now use a command similar to this to create the serial device if it does not exist:

```
/etc/mknod /dev/ttyXX c 1 0xSCf004
```

where /dev/ttyXX is a unique device, SC is the select code, and f is the function number.

4 To change the permissions on the device file, use the following command:

```
chmod 666 /dev/ttyXX
```

See also Consult your system documentation or ask an experienced system administrator for help with setting up a serial device.

To verify the low-level connection to the target system

If you are sure that your workstation can communicate with the monitor running on the target system, you can skip this step.

- Connect the target system to your host workstation via a serial cable.
- Use the MRI Communication Test Tool (**ctt68k**) to verify communication between your host workstation and your target.

  When you are finished, you should be able to connect with **cu** or **kermit** from your workstation to the MRI monitor and you should get the MRI monitor prompt "!@A" when you press the <**Return**> key.

  If you have problems with this connection, see the section "If you have problems" on page 26.

See also Refer to the MRI monitor documentation for detailed information on establishing communication with the target system.

To set up the host workstation files

1 Copy the support file and monitor executable file that you generated to a safe location on your host workstation.

If you built the monitor executable file on another workstation, and you use **ftp** to transfer the files, remember to set **binary** transfer for the binary files and **ascii** transfer for the ASCII files.

2 Set up the 64700tab.net file.

The 64700tab.net file is used by the mon64700 to define both the serial connection to the target and socket connection to the Debug Environment. The 64700tab.net

file also defines the type of HP 64700-compatible ROM monitor that mon64700 should present to the Debug Environment.

The 64700tab.net file is located in the directory $HP64000/etc (/usr/hp64000/etc by default).

For each MRI monitor/target that you have connected to a serial port on your host add a line to 64700tab.net. The line should have this format:

```
lan: <logical_name> <processor_type> <host_name> <port_number> <support_file>
```

<logical_name> The name that you want to attach to this monitor/target. This name can be any easy to remember name, perhaps a unique name for the target or processor (m68k).

<processor_type> A general classification of the processor type. Its value can be one of the following: m68000, m68020, m68030, m68040, m68302, m6833x, m68340, or m68360. If you are using an M6833x-series processor (for example, an M68331 or M68332), the processor_type must be "6833x"; use the T: line in the support file to specify the exact processor type (see page 8).

<host_name> The logical_name or the IP number of the workstation host where the the serial ports are connected.

<port_number> The designated socket port to be used when Debug Environment connects to a specific mon64700. Port numbers must be even numbers and must be unique for each mon64700/serial port pair used on a given workstation host. Typically 6470 is used for the first serial port and 6472 is used for the next, and so on. You may need to check with your system administrator to verify that the socket values that you select are not being used by any other process.

If you have multiple ROM monitors on multiple workstations, you can have a 6470 port on each workstation. If more than one ROM monitor is connected to a single workstation, you must designate unique socket port numbers (for example, 6470 and 6472).

<support_file> The full path file name of the support file that describes the serial port and how to start the ROM monitor.

Example Here is an example line from /usr/hp64000/etc/64700tab.net for a 68332 processor connected to workstation "lab2":

```
lan:  mon68332  m6833x  lab2  6470 /users/myproject/mon64700/xhm68k.sup
```

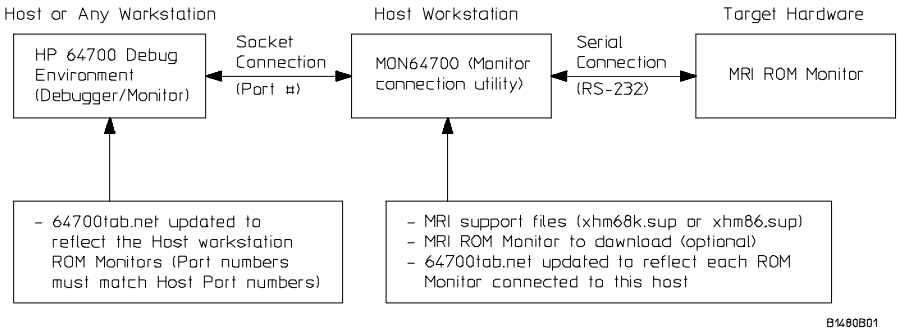
To set up additional workstations

You can run the Debugger/Monitor on a workstation other than the one connected to the ROM monitor.

- On additional hosts where you want to run the Debugger/Monitor add a line to 64700tab.net:

```
lan: <logical_name> <processor_type> <host_name> <port_number>
```

<port_number> The designated socket port to be used when Debug Environment connects to a specific mon64700. This port number must match the port number that you assigned to mon64700 on the host workstation where the actual serial connection is made. If you assigned 6470 as the port number on your host workstation, you could leave this field blank because 6470 will be used as default.



The 64700tab.net file is used by the Debug Environment to define the socket connection that should be used to find the specified HP64700 compatible ROM monitor. If you are running the Debug Environment on the same host as mon64700, 64700tab.net is already set up to communicate with the Debug Environment.

Using MON64700

To start mon64700

1 Verify that you have power to your monitor/target and that the serial line is still connected.

2 Enter the following command at the operating system prompt:

```
mon64700 <logical_name>
```

You should see the following message:

```
mon64700: Ready for Debugger/Monitor connection.
```

If you did not specify the processor type in your 64700tab.net file or in the support file, you will be prompted to enter a precise description of the processor type. You can avoid this question by adding a T:<processor_type> to your support file. The value of <processor_type> can be one of the following: m68000, m68020, m68030, m68040, m68302, m68331, m68332, m68340, or m68360.

If mon64700 does not find a support file associated with the logical_name, you will be prompted to enter the name of the support file. To avoid this question, be sure to add the name of a support file to the line describing your ROM monitor in the 64700tab.net file.

Example

If you have set up a monitor connection to an M68332 processor in your 64700tab.net file called "mon68332", enter:

```
mon64700 mon68332
```

To stop mon64700

1. Exit any debugger interfaces which are using the monitor.
2. Enter the following command at the operating system prompt:

```
mon64700 -q <logical_name>
```

Mon64700 is similar to the power switch on an HP64700 emulator. It is NOT intended to be shut down unless a major failure occurs, or you are moving the target to another host machine. You do not need to shut down mon64700 after you exit a Debug Environment session.

To verify that you can connect to your target

You can verify that mon64700 is working properly by means of an emul700 command.

- Enter the following command:

```
emul700 -lv <host_name>
```

where <host_name> is your logical host name or IP address where the monitor/target is connected via the serial line. You should obtain a status report like the following:

```
m68k - m68020 available
description:     M68020 ROM Monitor
user interfaces:  xdebugmon
internet address: <host_name>
```

The emul700 command is usually used to start an emulator/analyzer graphical interface. If you specify a ROM monitor system instead of an emulator, emul700 will start the Debugger/Monitor interface. The "-lv" option checks the status of the emulator or mon64700 and then exits without starting the interface.

To start a Debug Environment connection to the target

Note

Remember to start mon64700 before you start any of the Debug Environment interfaces.

- To use the debugger interface, enter:

```
db68k -e <logical_name> &
```

- To use the debugger interface, you may also enter:

```
emul700 <logical_name> &
```

- To use the HP SoftBench interface, enter:

```
debug64700 <logical_name> &
```

If you only have one ROM monitor connected to a workstation, you can use the host name of the workstation instead of <logical_name> in all of the above commands.

See also

The debugger *User's Guide*
emul700(1)
debug64700(1)
db68k(1)
db86(1)

To compile a program

There are a few constraints that you need to be aware of when you compile your program for the Debugger/Monitor:

- Be aware of the memory location of your monitor. Your final link should not try to load code on top of the monitor code or the monitor data locations. The starting address and ending address of the monitor program are displayed when you start mon64700.
- Avoid using the vectors that the MRI monitor uses to communicate and run. By default, mon64700 is configured to protect the breakpoint vector and the trace vector (the trace vector is used for stepping). All other vectors are unprotected. For example, if your i8086 monitor uses the NMI vector and your target executable file overwrites this vector, the monitor connection will hang or terminate.

  Emulators (and some other "execution engines") require the initialization of some of the vectors which are needed by the ROM monitor. You can use the same executable file in both the emulator and the ROM monitor if you protect these vectors from being overwritten. The Debugger/Monitor can automatically protect a range of vectors. Select **Modify→Config** from the Debugger/Monitor menu bar to define the range of vectors to protect.
- The initial value of the interrupt mask is set upon reset to an initial state. The first instructions of your program should change the interrupt mask to an appropriate value so that the break command will work properly. For example, you could set the interrupt mask to 0.
- If your processor has reset vectors, they should be defined to appropriate values. Generally speaking the reset vectors should be compiled in to your target executable.

To modify a Debug Environment demo to work with your ROM monitor

A simple demonstration program has been provided. With two modifications to a linker command file, this demo should work with your monitor.

1 Find the proper Debug Environment demo directory:

```
Processor       Debug Environment demo directory
M68000         $HP64000/demo/debug_env/hp64742
M68020         $HP64000/demo/debug_env/hp64748
M68030         $HP64000/demo/debug_env/hp64747
```

```
M68040          $HP64000/demo/debug_env/hp64783
M68302          $HP64000/demo/debug_env/hp64746
M6833x          $HP64000/demo/debug_env/hp64749
M68340          $HP64000/demo/debug_env/hp64751
M68360          $HP64000/demo/debug_env/hp64780
```

where $HP64000=/usr/hp64000 by default.

2 Copy the demo files to a new directory. For example, if you are using an M68020:

```
cd $HOME
cp -r $HP64000/demo/debug_env/hp64748 hp64748mon
```

Using the HP AxLS compiler

- Edit the file Linkmon.k and verify that the program, data, and stack portions of the program will be in available RAM memory. In particular, check the values on these lines:

```
SECT  env=$6000          ; Load address for prog/const sections
SECT  envdata=$B000      ; Load address for data sections
SECT  stack=$C200        ; Load address for stack section
```

- Remove the old object files with the **make clean** command.

- Compile the demo program with **make ecsmon**

Using the MRI compiler

- If you have not already done so, set and export the MRI_68K_BIN environment variable.

- Edit the file mri/linkmon.cmd and verify that the program, data, and stack portions of the program will be in available RAM memory. In particular check the values on these lines:

```
common   stack=$0C200   ; set starting address of stack section
sect     literals=$6000 ; set starting address of prog/const sections
sect     vars=$B000     ; set starting address of data sections
```

- Remove the old object files with the **make clean** command.
- Compile the demo program with **make ecsmon.mri**

To run the Debug Environment demo

- Modify the demo program as described in the previous section.
- Start the demo program as described in the debugger *User's Guide.*
- Before you execute the demo program, choose **Execution→Reset to Monitor**.

To load a program

- In the debugger interface, select **File→Load→Executable** and enter the name of the executable to load.

  If you are using the Debug Environment demo, enter "ecsmon.x".

  Downloading over the serial port will take longer than downloading into an emulator. Larger programs may take a considerable amount of time (especially at 9600 baud). To improve the download rate, you may consider increasing the ROM monitor baud rate, or you may consider alternative methods of getting your program into your target and then only loading the symbols in the Debugger/Monitor.

To run a program

1 Verify that the stack pointer is pointing to a usable RAM location.

If you have compiled the Debug Environment Demo, you can select **Execution→Reset to Monitor** to correctly initialize the stack pointer and program counter.

2 Use the step, set breakpoint and run commands just as you use them in the Debugger/Simulator or the Debugger/Emulator.

See also Refer to the debugger manual for complete details about running programs in the debugger interface.

To stop program execution

Here are two ways to stop a running program:

- With the mouse pointer in the main debugger window, press <Ctrl>-C.

If you have specified an interrupt (or other mechanism) to cause a break, you can use <Ctrl>-C to stop program execution.

Or

- Press the abort button on your target.

See also If the program does not stop, see page 29.

To connect to HP SoftBench

1. Compile your programs for static analysis.
2. Set X resources for the SoftBench action keys.
3. Verify that SoftBench is properly configured.
4. Start the Debug64700 system.

See also

See the *Debug64700 User's Guide* for detailed information on setting up your project for Debug64700.

To configure the debugger

- In the debugger interface, select **Modify→Configuration...**.

  The items you can configure vary from processor to processor. In most cases, you can configure the following items:

  - Protect Breakpoint and Step vectors?
  - Protect vectors 0h thru memory address?

  Details about the configuration items are provided in the on-line help in the configuration dialog.

Configuration files

You can save the results of your configuration session to a file. Select **File→Save...** or **Store→Configuration**... from the configuration dialog. Enter a configuration file. The name of the file will be appended with ".MA" (Debugger/Monitor ASCII configuration file).

See also For information on using the configuration dialog, refer to the "Configuring the Emulator" chapter in the debugger *User's Guide*.

To end the debugger connection

- In the debugger interface, select **File→Exit→Released**.

  This will exit the Debugger/Monitor and allow another user access to the target system.

  Exiting the Debugger/Monitor does not affect operation of mon64700. You do not need to stop mon64700 at this point unless you plan to physically disconnect the target system from the host workstation.

To check the status of mon64700

- At the operating system prompt, enter:

  ```
  mon64700 -s <logical_name>
  ```

  The status of the ROM monitor connection will be displayed.

To continuously monitor the ROM monitor connection

- At the operating system prompt, enter:

  ```
  mon64700 -l <logical_name>
  ```

This command displays messages that can tell you whether the connection has been started, interrupted, or reestablished. This command runs in the foreground and will continue to write error messages to the window in which it was started. It may be best to start this command in its own separate window and iconify it after it is started. This command does not display any of the data which is exchanged between the debugger interface and the target system.

To define a new reset function

You can write a new reset function and make it appear on an action key:

1 Decide which debugger commands your reset function needs to perform.

2 Define a reset macro in the debugger startup file.

3 Define an action key to execute the reset macro.

Why you might need to define a new reset function

The Debugger/Monitor uses a pseudo processor reset function which differs from the reset function used by an emulator or simulator. The pseudo reset does the following:

1 The status register is set to 0x2700.

2 A new stack pointer is loaded from vector 0.

3 A new program counter is loaded from vector 1.

4 The monitor is entered.

The pseudo reset does *not* modify the following registers:

- The vector base register (VBR).
- The cache control registers.
- The address or data registers.

If the serial communication peripheral is reset, the communication link between the target and the debugger interface will be broken. For this reason a RST is not executed when doing a reset from the Debugger/Monitor interface.

If the reset function provided does not meet your needs, you can write your own reset function.

Example

The following macro resets the VBR register then performs the same actions as the default reset function:

```
/* reset macro for Motorola 68030, 68040, and CPU32 processors */
D M A  my_reset()
{
   long *ptr;

   @SR=0x2700;
   @VBR=0;
   /* set stack pointer from vector 0, address is (VBR+(4*0))=0 */
   ptr=0;
   @A7=*ptr;
   /* set stack pointer from vector 0, address is (VBR+(4*1))=4 */
   ptr=4;
   @PC=*ptr;
}
.
```

To use this macro with the debugger demonstration program: add the macro to the end of the Cmd_dbmac.com file, which you can find under cmdfiles/debug in your demo directory. Notice that you must finish the macro with a period on a line by itself.

To add the macro to one of the demo program action keys, append the provided Xdefaults.all file to the .Xdefaults file in your home directory. Look for a "Your Key" line like the following:

```
debugmon.processor.actionKeysSub.keyDefs:
...
"< Your Key >"   "D H tellkeysHP     InBrowser" \
```

Change the "Your Key" line to:

```
"My_reset"       "D M C my_reset( )" \
```

Notice that the function call uses a space between the parentheses.

When you start the demo program with **Startdebug**, the new action key will appear. If you have not defined the XENVIRONMENT environment variable, you may need to restart your window manager before this change will take effect.

How mon64700 works

This section provides some additional information on how mon64700 works.

How mon64700 determines the processor type

Mon64700 looks in the following places to determine the type of processor in the target system:

1 The processor type specified in the 64700tab.net file.

2 The processor type specified on the T: line in the support file.

3 If the processor type is not specified in the 64700tab.net file or the support file, then when it is started, mon64700 will prompt for the processor type.

Base Address Registers (CPU32 and M68302)

The Motorola M68302 and CPU32 processors provide on-chip peripheral devices. The registers associated with these devices are mapped into a contiguous block of memory starting at some base address.

The Debugger/Monitor does not make use of the base address information stored by the processor. You must select the base address via the Debugger/Monitor configuration process or register modification commands.

The form of the base address depends upon the processor type.

M68302

The M68302 processor stores the upper 12 bits of the base address in bits 0 through 11 of memory location $0F2. The Debugger/Monitor ignores this value.

Use the M68302 Debugger/Monitor configuration dialog to specify a value for the upper 12 bits of the base address used by the debugger. The BAR register shown by the Debugger/Monitor reflects this value. The actual value of the processor BAR register may be examined by displaying the memory at location $0F2.

M6833x

The Motorola M6833x family of processors uses the state of the module configuration register (the MCR, duplicated as the SIM_MCR) modmap bit (MM) to determine the location of the peripheral devices. The MCR register is located in the system integration module which is mapped by the value of the MCR MM bit. In other words, the location of the MCR register depends on the value of the MCR register.

The M6833x Debugger/Monitor configuration allows you to select one of two possible values for the base address. The value shown by the debugger for the MCR is read from the MCR register on the processor.

M6834x and M6836x

The value of the base address is stored in the base address register (MBAR). This register can be accessed by using the MOVES instruction to address $0003FF00 in the CPU function code address space. The Debugger/Monitor ignores this value.

You must specify a value to be used as the MBAR for locating memory mapped device registers. Debugger reads and writes to MBAR act on a copy of MBAR which is local to the debugger; debugger reads and writes have no effect on the actual processor MBAR register.

If you have problems

The following sections describe some common problems and what to do about them.

If you have problems setting up or communicating with the monitor, you should also consult the MRI monitor documentation.

If you have problems using the debugger, you should also consult your HP debugger *User's Guide*.

If your target system is not communicating with your workstation

- ☐ Check that you are using the proper serial port name. Examples of typical serial device names are: /dev/plt_rs232_a (for HP 9000 series 700 workstations), /dev/ttya (for Sun workstations), and /dev/tty00 (for HP 9000 series 300 workstations).

- ☐ If the serial port device file does not exist, you need to create one as described on page 9.

- ☐ Check that the serial device file has open permissions.

  Use an **ls** command to verify that the permissions are "read or write by anyone." The permissions should look like this:

  ```
  $ ls -l  /dev/plt_rs232_a
  crw-rw-rw-  2 lp   bin    1 0x204004 Jan 15  1994 /dev/plt_rs232_a
  ```

  To change permissions to allow the file to be read or written by anyone, enter a command like this:

  ```
  chmod 666 /dev/plt_rs232_a
  ```

- ☐ Check that you are using the appropriate serial cable. Some hosts need a null modem; others need a direct cable.

☐ If you are using **cu** to communicate with the target system, check that you have set up the device name.

For example, on an HP 9000 Series 700 host, check that the following lines are in /usr/lib/uucp/Devices:

```
Direct plt_rs232_a - 9600 direct
Direct plt_rs232_b - 9600 direct
```

If they are not found, add them using root privileges.

☐ Verify that you can make a low-level connection to the target system.

Make the connection to the serial port. Use a command like:

```
/usr/bin/cu -l /dev/plt_rs232_a
```

Press the <Return> key. You should see a monitor prompt (for example, "CPU32Bug" or "!@A"). If not, check power and the serial cable.

To exit **cu**, press enter ~. (a tilde followed by a period).

If **kermit** is available on your system, you may find it easier to use **kermit** than cu. Enter the following commands:

```
kermit
set line <device file name>
set baud <baud rate>
connect
```

☐ Check that your downloadable monitor is properly built. Refer to your MRI manuals for complete details.

If the connection has been interrupted

If, for example, the serial cable has been disconnected then reconnected, the monitor connection should be reestablished automatically. Select **Settings→Assembly Level Debug** and then **Settings→High Level Debug** to view the current value of the PC. You can watch what is happening to the connection by using the **mon64700 -l** command.

If the debugger interface stops responding

If communications are interrupted during a program download, the debugger interface may stop responding to keyboard or mouse commands. If this happens:

1 Use the **ps -e** command to find the process ID numbers of the debugger (xdb*xxx*), emul700dmn, and mon64700.
2 Use the **kill** command to stop these processes.
3 Reset the target system.
4 Restart mon64700 and the debugger.

If the workstation "hangs"

On some workstations, toggling power to the target system (whether or not mon64700 is running) can cause the workstation to "hang" (stop working altogether). This is a problem with the way the workstation handles serial communications. If this happens, try any of the following:

☐ Disconnect then reconnect the serial cable.

☐ Toggle power to the target system a few times.

☐ Turn off power to the workstation and the target system.

If the target ROM monitor stops operating properly

☐ Try pressing the abort button, if any, on your target system.

☐ Press the reset button (if any).

After pressing the reset button you may have to restart mon64700 to get resynchronized. Use **mon64700 -q <logical_name>** to halt mon64700 and **mon64700 <logical_name>** to restart mon64700.

If the Debugger/Monitor fails to start properly

This problem can occur when an old version of the B1471 Operating Environment software has been installed over the new version. Check that the version of the B1471 software is A.05.30 or higher (A.06.10 or higher for Solaris), then re-install B1471 from the debugger tape if necessary.

If the debugger takes a long time to start

- ☐ Determine if the debugger startup file is downloading a large executable file to the target. If the executable doesn't need to be downloaded every time the debugger starts, edit the startup file (for Motorola processors, the file name is db68k.rc by default).

If the program does not stop

Normally, you can stop a running program by pressing <Ctrl>-C or by pressing an "abort" button on the target system. If the program doesn't stop, check for these conditions:

- ☐ The interrupt mask may be set to exclude the level of interrupt you are using to break program execution.

- ☐ The monitor may have been written to do polled I/O. The polling loop may not be interruptible.

- ☐ The program in the target system may have entered a state which has disabled interrupts or corrupted the monitor.

If mon64700 cannot connect

If you see the following error messages:

```
mon64700: Unable to connect to socket port 6472
mon64700: Is "mon64700 mon68332" already running?
mon64700: Is an emul700dmn or other debugger process still running?
mon64700: mon68332 Quitting
```

- ☐ Check that mon64700 is not already running. Use **mon64700 -s <*logical_name*>** to determine whether mon64700 is running. To restart mon64700, use **mon64700 -q <*logical_name*>** to halt mon64700. Then use **mon64700 <*logical_name*>** to restart.

- ☐ Check whether an old emul700dmn or db68k process is still running from an aborted session. You will have to kill these processes before you can start mon64700.

- ☐ If you have just aborted mon64700, wait 1-2 minutes and try starting mon64700 again. Some workstations take a few minutes to clean up sockets from an aborted session.