User's Guide

Real-Time C Debugger for 68302

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

MS-DOS(R) is a U.S. registered trademark of Microsoft Corporation.

HP-UX 9.* and 10.0 for HP 9000 Series 700 and 800 computers are X/Open Company UNIX 93 branded products.

TrueType(TM) is a U.S. trademark of Apple Computer, Inc.

UNIX(R) is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Windows or MS Windows is a U.S. trademark of Microsoft Corporation.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	B3621-97000, August 1993
Edition 2	B3621-97001, April 1994
Edition 3	B3621-97002, June 1995
Edition 4	B3621-97003, August 1995

Safety, Certification and Warranty

Safety and certification and warranty information can be found at the end of this manual on the pages before the back cover.

Real-Time C Debugger — Overview

The Real-Time C Debugger is an MS Windows application that lets you debug C language programs for embedded microprocessor systems.

The debugger controls HP 64700 emulators and analyzers either on the local area network (LAN) or connected to a personal computer with an RS-232C interface or the HP 64037 RS-422 interface. It takes full advantage of the emulator's real-time capabilities to allow effective debug of C programs while running in real-time.

The debugger is an MS Windows application

- You can display different types of debugger information in different windows, just as you display other windows in MS Windows applications.
- You can complete a wide variety of debug-related tasks without exiting the debugger. You can, for example, edit files or compile your programs without exiting the debugger.
- You can cut text from the debugger windows to the clipboard, and clipboard contents may be pasted into other windows or dialog boxes.

The debugger communicates at high speeds

- You can use the HP 64700 LAN connection or the RS-422 connection for high-speed data transfer (including program download). These connections give you an efficient debugging environment.

You can debug programs in C context

- You can display C language source files (optionally with intermixed assembly language code).
- You can display program symbols.
- You can display the stack backtrace.
- You can display and edit the contents of program variables.
- You can step through programs, either by source lines or assembly language instructions.
- You can step over functions.
- You can run programs until the current function returns.
- You can run programs up to a particular source line or assembly language instruction.

- You can set breakpoints in the program and define macros (which are collections of debugger commands) that execute when the breakpoint is hit. Break macros provide for effective debugging without repeated command entry.

You can display and modify processor resources

- You can display and edit the contents of memory locations in hexadecimal or as C variables.
- You can display and edit the contents of microprocessor registers including on-chip peripheral registers.
- You can display and modify individual bits and fields of bit-oriented registers.

You can trace program execution

- You can trace control flow at the C function level.
- You can trace the callers of a function.
- You can trace control flow within a function at the C statement level.
- You can trace all C statements that access a variable.
- You can trace before, and break program execution on, a C variable being set to a specified value.
- You can make custom trace specifications.

You can debug your program while it runs continuously at full speed

- You can configure the debugger to prevent it from automatically initiating any action that may interrupt user program execution. This ensures that the user program executes in real time, so you can debug your design while it runs in a real-world operating mode.
- You can inspect and modify C variables and data structures without interrupting execution.
- You can set and clear breakpoints without interrupting execution.
- You can perform all logic analysis functions, observing C program and variable activity, without interrupting program execution.

In This Book

This book documents the Real-Time C Debugger for 68302. It is organized into five parts whose chapters are described below.

Part 1. Quick Start Guide

Chapter 1 quickly shows you how to use the debugger.

Part 2. User's Guide

Chapter 2 shows you how to use the debugger interface.
Chapter 3 shows you how to configure the emulator.
Chapter 4 shows you how to plug the emulator into target systems.
Chapter 5 shows how to perform the tasks that you can use to debug programs.

Part 3. Reference

Chapter 6 contains a summary of the debugger commands as they are used in command files and break macros.
Chapter 7 describes the format for expressions used in commands.
Chapter 8 describes commands that appear in the menu bar.
Chapter 9 describes commands that appear in debugger window control menus.
Chapter 10 describes commands that appear in popup menus.
Chapter 11 describes commands that are only available in command files and break macros.
Chapter 12 describes error messages and provides recovery information.

Part 4. Concept Guide

Chapter 13 contains conceptual (and more detailed) information on various topics.

Part 5. Installation Guide

Chapter 14 shows you how to install the debugger.
Chapter 15 shows you how to install or update HP 64700 firmware.

Contents

Part 2 User's Guide

2 Using the Debugger Interface

3 Configuring the Emulator

4 Plugging the Emulator into Target Systems

5 Debugging Programs

Part 3 Reference

6 Command File and Macro Command Summary

7 Expressions in Commands

8 Menu Bar Commands

11 Other Command File and Macro Commands

12 Error Messages

Part 4 Concept Guide

13 Concepts

Part 5 Installation Guide

Part 1

Quick Start Guide

A few task instructions to help you get comfortable.

1

Getting Started

Getting Started

This tutorial helps you get comfortable by showing you how to perform some measurements on a demo program. This tutorial shows you how to:

1 Start the debugger.
2 Adjust the fonts and window size.
3 Set the reset value for the supervisor stack pointer.
4 Map memory for the demo program.
5 Load the demo program.
6 Display the source file.
7 Set a breakpoint.
8 Run the demo program.
9 Delete the breakpoint.
10 Single-step one line.
11 Single-step 10 lines.
12 Display a variable.
13 Edit a variable.
14 Monitor a variable in the WatchPoint window.
15 Run until return from current function.
16 Step over a function.
17 Run the program to a specified line.
18 Display register contents.
19 Trace function flow.
20 Trace a function's callers.
21 Trace access to a variable.
22 Exit the debugger.

Demo Programs

Demo programs are included with the Real-Time C Debugger in the C:\HP\RTC\M302\DEMO directory (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software).

Subdirectories exist for the SAMPLE demo program, which is a simple C program that does case conversion on a couple strings, and for the ECS demo program, which is a somewhat more complex C program for an environmental control system.

Each of these demo program directories contains a README file that describes the program and batch files that show you how the object files were made.

This tutorial shows you how to perform some measurements on the SAMPLE demo program.

Step 1. Start the debugger

- Open the HP Real-Time C Debugger group box and double-click the 68302 debugger icon.

  Or:

1. Choose the File→Run (ALT, F, R) command in the Windows Program Manager.
2. Enter the debugger startup command, C:\HP\RTC\M302\B3621.EXE (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software).
3. Choose the OK button.

Step 2. Adjust the fonts and window size

The first time RTC is used, a default window and font size is used. This may not be the best for your display. You may change the font type and size with the Settings→Font... command, and change the window size by using standard Windows 3.1 methods (moving the mouse to the edge of the window and dragging the mouse to resize the window).

1 Choose the Settings→Font... (ALT, S, F) command.

2 Choose the Font, Font Style, and Size desired in the Font dialog box.

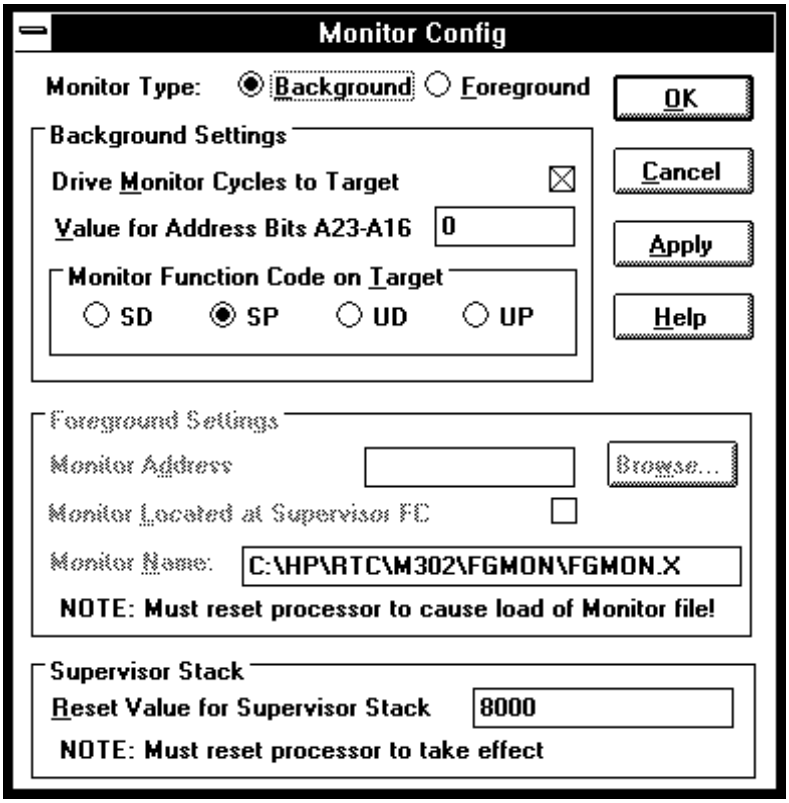
3 Choose the OK button to apply your font selections and close the Font dialog box.

The sizes of the RTC window, as well as the sizes of the windows within RTC, and the fonts used will be saved in the B3621.INI file and reused when you enter RTC the next time.

Step 3. Set the reset value for the supervisor stack pointer

1 Choose the Settings→Emulator Config→Monitor... (ALT, S, E, O) command.

2 Enter "8000" in the Reset Value for Supervisor Stack text box.



3 Choose the OK button.

The 68302 emulator requires the supervisor stack pointer to be set to an even address in emulation RAM or in target system RAM.

When you break the emulation processor from the EMULATION RESET state into the RUNNING IN MONITOR state, the supervisor stack pointer is set to the address specified.

You can also set the supervisor stack pointer by modifying the SSP register in the Register window.

Note

Breaking into the monitor from a state other than EMULATION RESET does not cause the supervisor stack pointer to be modified. (The Execution→Reset (ALT, E, E) command places the emulator in the EMULATION RESET state.)

Step 4. Map memory for the demo program

By default, the emulator assumes all memory addresses are in RAM space in your target system. If you wish to load some of your target program in emulation memory, or identify some of your memory addresses as ROM or Guarded, those specifications must be entered in the memory map.

The demo program reserves addresses 0h-0fffh for ROM and 6000h-0ffffh for RAM. Map these address ranges as emulation memory.

Because the BAR and SCR registers are located in the exception vector table (at 0F0H and 0F4H), the exception vector table (0 through 3FFH) should be mapped as emulation RAM.

The internal memory space of the 68302 must be mapped as target RAM.

1 Choose the Settings→Emulator Config→Memory Map... (ALT, S, E, M) command.

2 Enter "0" in the Start text box.

3 Tab the cursor to the End text box and enter "3ff".

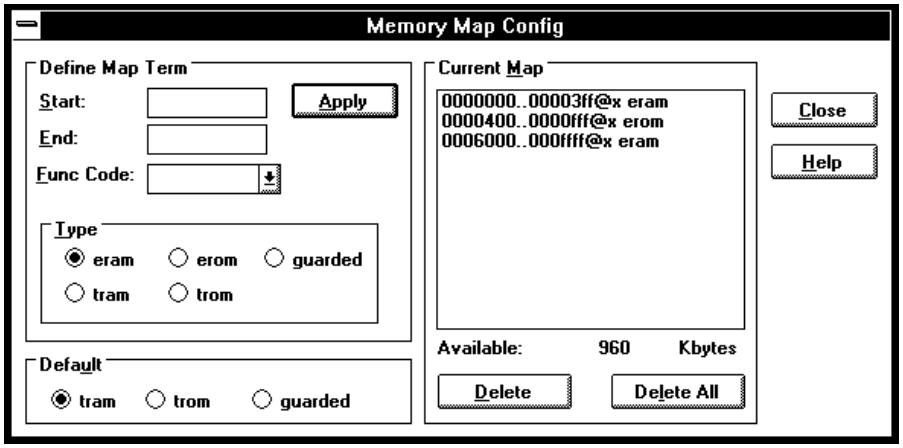
4 Select "eram" in the Type option box.

5 Choose the Apply button.

6 Enter "400" in the Start text box, enter "0fff" in the End text box, select "erom" in the Type option box, and choose the Apply button.

7 Enter "6000" in the Start text box and "0ffff" in the End text box, select "eram" in the Type option box, and choose the Apply button.

8 Select "tram" in the Default group box. This maps all memory ranges not listed in the Current Map as target system RAM. The 68302 internal memory space is not listed in the Current Map; therefore, it is mapped as target system RAM.



9 Choose the Close button.

Step 5. Load the demo program

1. Choose the File→Load Object... (ALT, F, L) command.
2. Choose the Browse button and select the sample program object file, C:\HP\RTC\M302\DEMO\SAMPLE\SAMPLE.X (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software).
3. Choose the OK button in the Object File Name dialog box.
4. Choose the Load button.

Step 6. Display the source file

To display the sample.c source file starting from the main function:

1. If the Source window is not open, double-click on the Source window icon to open the window. Or, choose the Window→Source command.

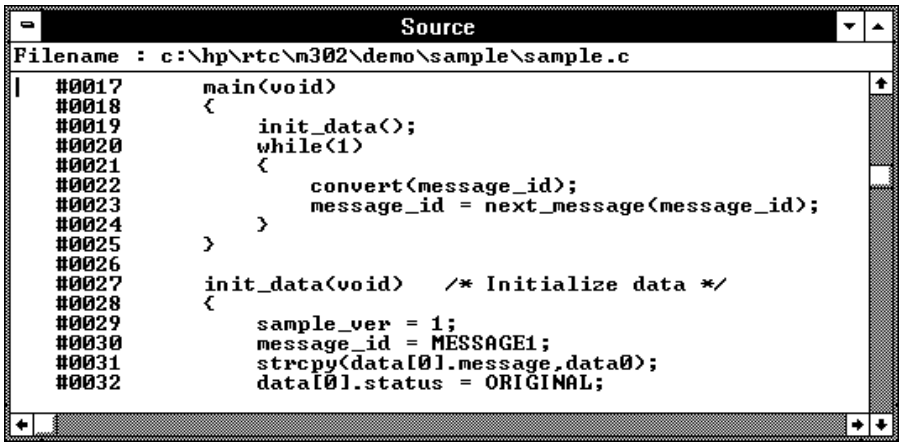
2. From the Source window's *control menu*, choose Search→Function... (ALT, -, R, F) command.

3. Select "main".

4. Choose the Find button.

5. Choose the Close button.

6. From the Source window's *control menu*, choose Display→Source Only (ALT, -, D, S) command.



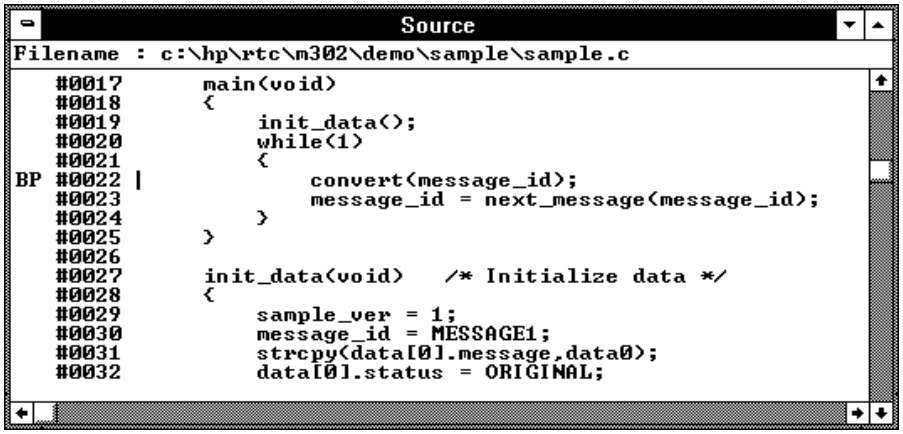
The window displays sample.c source file, starting from main function.

Step 7. Set a breakpoint

To set a breakpoint on line 22 in sample.c:

1. Cursor-select line 22 (that is, move the mouse pointer over line 22 and click the left mouse button).

2. Choose the Breakpoint→Set at Cursor (ALT, B, S) command.



Notice that line 22 is marked with "BP" which indicates a breakpoint has been set on the line.

Note

This can be done more quickly by using the pop-up menu available with the right mouse button.

Step 8. Run the demo program

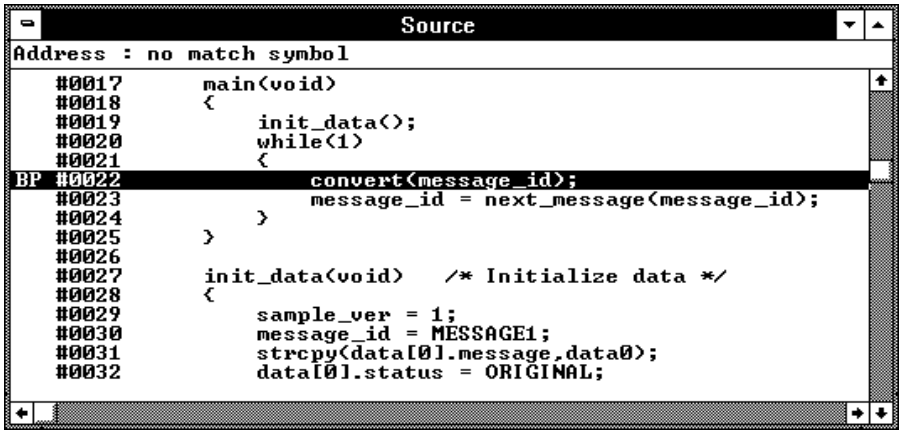
To run the demo program from the transfer address:

1. Choose the Execution→Reset (ALT, E, E) command followed by the Execution→Break (ALT, E, B) command to initialize the supervisor stack pointer.

2. Choose the Execution→Run... (ALT, E, R) command.

3. Select the Start Address option.

4. Choose the Run button.



Notice the demo program runs until line 22. The highlighted line indicates the current program counter.

Step 9. Delete the breakpoint

To delete the breakpoint set on line 22:

1. Cursor-select line 22.
2. Choose the Breakpoint→Delete at Cursor (ALT, B, D) command.

   The "BP" marker disappears in the Source window.

Step 10. Single-step one line

To single-step the demo program from the current program counter:

- Choose the Execution→Single Step (ALT, E, N) command. Or, press the F2 key.

  Notice the C statement executed and the program counter is at the "convert" function.

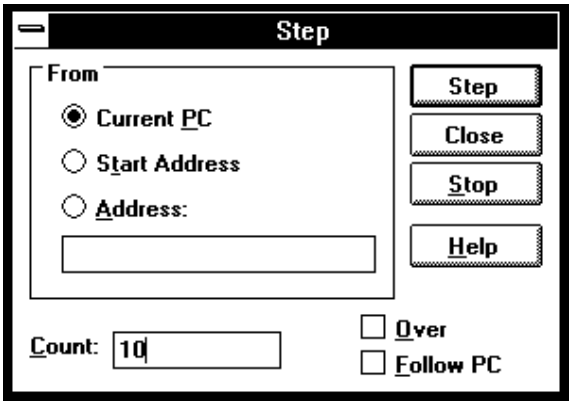
Step 11. Single-step 10 lines

To single-step 10 consecutive executable statements from the current PC line:

1 Choose the Execution→Step... (ALT, E, S) command.

2 Select the Current PC option.

3 Enter "10" in the Count text box.



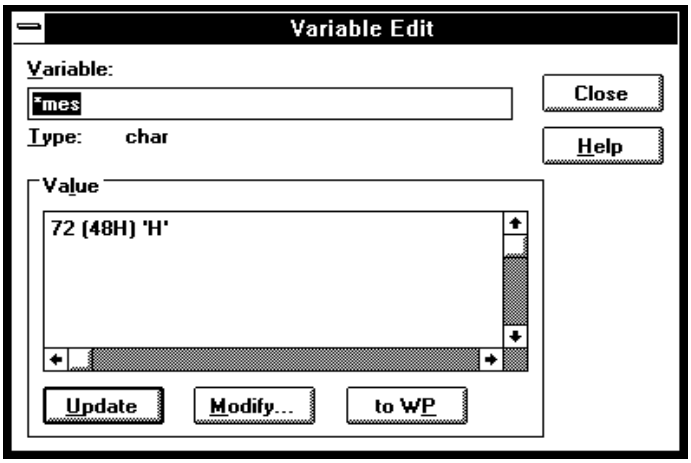
4 Choose the Step button. Notice that the step count decrements by one as the program executes step by step. The step count stops at 1.

5 Choose the Close button.

Step 12. Display a variable

To display the contents of auto variable "*mes":

1. Drag "*mes" on line 45 in the Source window until it is highlighted.
2. Choose the Variable→Edit... (ALT, V, E) command.



The Variable text box displays "*mes".

Notice the Value list box displays the contents of "*mes".

Note

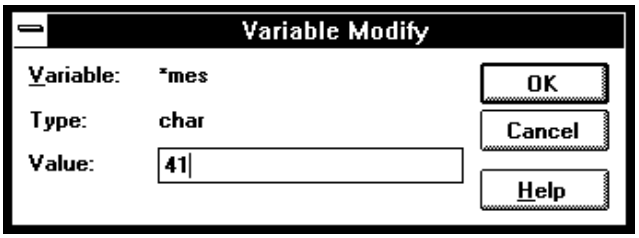
You can only register or display an auto variable as a watchpoint while the program counter is within the function in which the variable name is declared.

Step 13. Edit a variable

To edit the contents of variable "*mes":

1 In the Variable Edit dialog box, choose the Modify button.

2 Enter "41" in the Value text box.



3 Choose the OK button.

4 Notice the contents of the variable in the Value list box has changed to "41".

Step 14. Monitor a variable in the WatchPoint window

The WatchPoint window lets you define a set of variables that may be looked at and modified often. For these types of variables, using the WatchPoint window is more convenient than using the Variable→Edit... (ALT, V, E) command.

To monitor the variable "*mes" in the WatchPoint window:

1. In the Variable Edit dialog box, choose the "to WP" button.
2. Choose the Close button.
3. Choose the Window→WatchPoint command.

Notice the variable "*mes" has been registered as a watchpoint.

Step 15. Run until return from current function

To execute the program until "convert_case" (the current PC function) returns to its caller:

1 Choose the Execution→Run to Caller (ALT, E, T) command.

The program executes until the line that called "convert_case".

2 Choose the Execution→Single Step (ALT, E, N) command (or press the F2 key) to go to the line that follows the return from the "convert_case:" function.

Step 16. Step over a function

To step over "change_status":

- Choose the Execution→Step Over (ALT, E, O) command. Or, press the F3 key.

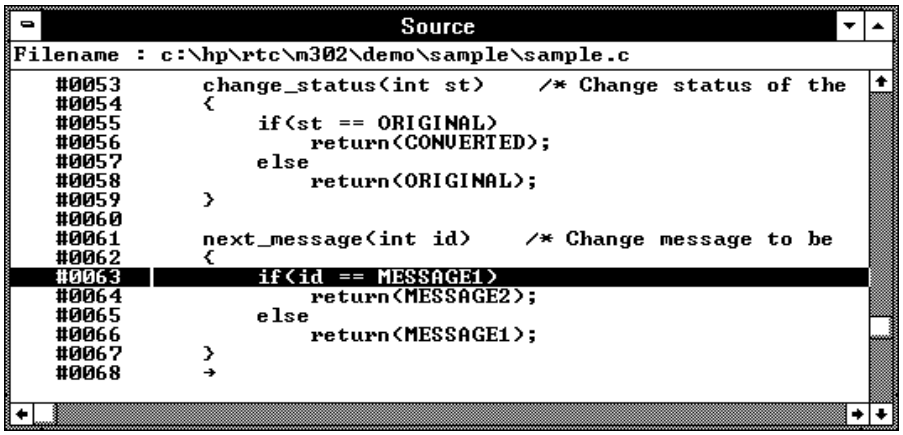
  The "change_status" function executes, and the program counter indicates line 41.

Step 17. Run the program to a specified line

To execute the demo program to the first line of "next_message":

1 Cursor-select line 63.

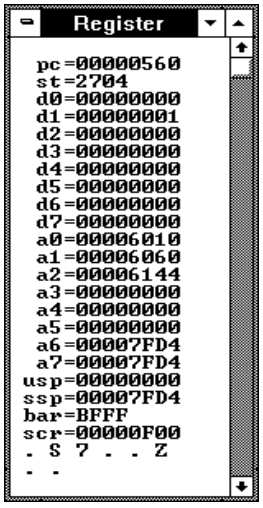
2 Choose the Execution→Run to Cursor (ALT, E, C) command.



The program executes and stops immediately before line 63.

Step 18. Display register contents

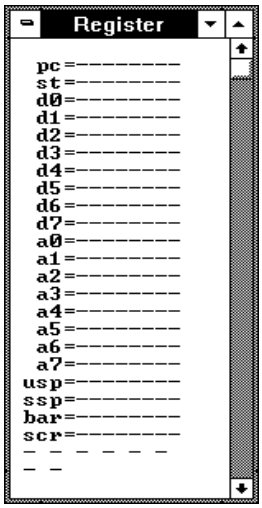
1 Choose the Window→Register command.



The Register window opens and displays the register contents. The display is updated periodically.

2 To see the effects of preventing monitor intrusion (running in real-time mode), choose the RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command.

3 To run the program, choose the Execution→Run (ALT, E, U) command. Or, press the F5 key.



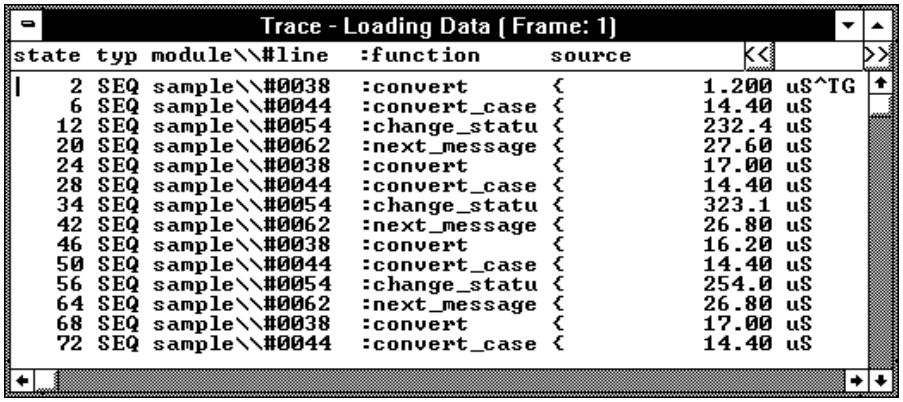
Notice that register contents are replaced with "----" in the display. This shows the debugger cannot update the register display. In order for the emulator to update its register display, the emulation monitor must interrupt target program execution while it reads the registers.

4 Choose the RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command to deselect the real-time mode. Notice that the contents of the registers are updated periodically.

Step 19. Trace function flow

- Choose the Trace→Function Flow (ALT, T, F) command.

The Trace window becomes active and displays execution flow as shown below.



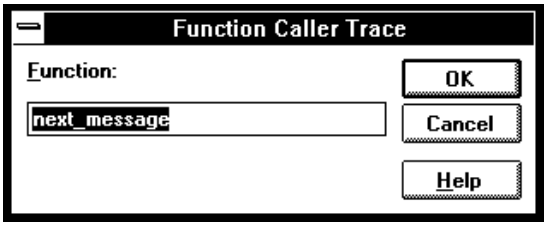
The command traces, and stores in trace memory, only the entry points to functions. This lets you check program execution flow.

Step 20. Trace a function's callers

To trace the caller of "next_message":

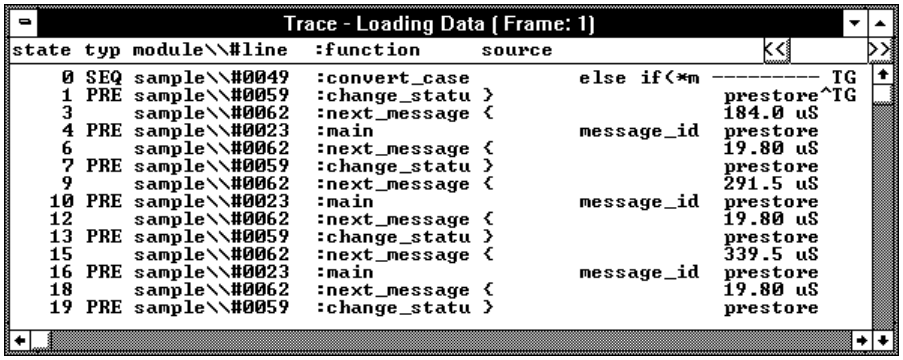
1 Double-click "next_message" in the Trace window or on line 61 in the Source window.

2 Choose the Trace→Function Caller... (ALT, T, C) command.



3 Choose the OK button.

The Trace window becomes active and displays the caller as shown below.



This command stores the first statement of a function and prestores statements that occur before the first statement (notice the state type PRE). The prestored statements show the caller of the function. In the above example, "next_message" is called by line 23 of "main". Because the first

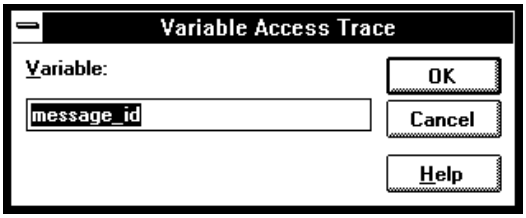
statement of "next_message" is prefetched after "change_status", these states are also included in the trace.

Step 21. Trace access to a variable

To trace access to variable "message_id":

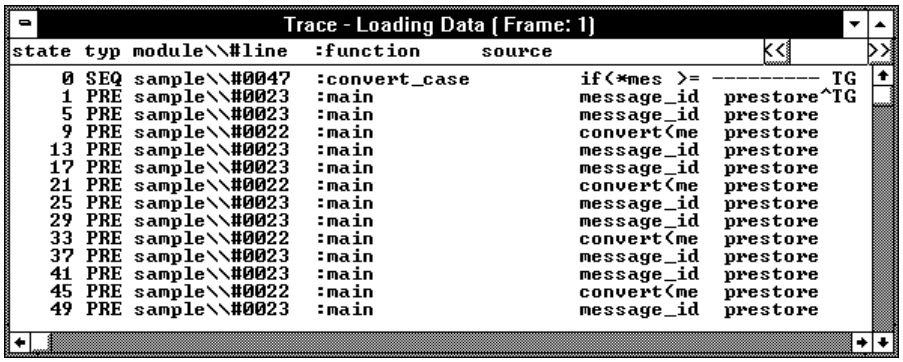
1 Double-click "message_id" in the Trace window or on line 22 in the Source window.

2 Choose the Trace→Variable Access... (ALT, T, V) command.



3 Choose the OK button.

The Trace window becomes active and displays accesses to "message_id" as shown below.



Line 23 displays twice because it accessed "message_id" twice for read and write.

Step 22. Exit the debugger

1. Choose the File→Exit (ALT, F, X) command.
2. Choose the OK button.

   This will end your Real-Time C Debugger session.

Part 2

User's Guide

A complete set of task instructions and problem-solving guidelines, with a few basic concepts.

2

Using the Debugger Interface

Using the Debugger Interface

This chapter contains general information about using the debugger interface.

- How the Debugger Uses the Clipboard
- Debugger Function Key Definitions
- Starting and Exiting the Debugger
- Working with Debugger Windows
- Using Command Files

How the Debugger Uses the Clipboard

Whenever something is selected with the standard windows double-click, it is placed on the clipboard. The clipboard can be pasted into selected fields by clicking the right mouse button.

Double-clicks are also used in the Register and Memory windows to make values active for editing. These double-clicks also copy the current value to the clipboard, destroying anything you might have wanted to paste into the window (for example, a symbol into the memory address field). In situations like this, you can press the CTRL key while double-clicking to prevent the selected value from being copied to the clipboard. This allows you to, for example, double-click on a symbol, CTRL+double-click to activate a register value for editing, and click the right mouse button to paste the symbol value into the register.

Many of the Real-Time C Debugger commands and their dialog boxes open with the clipboard contents automatically pasted in the dialog box. This makes entering commands easy. For example, when tracing accesses to a program variable, you can double-click on the variable name in one of the debugger windows, choose the Trace→Variable Access... (ALT, T, V) command, and click the OK button without having to enter or paste the variable name in the dialog box (since it is has automatically been pasted in the dialog box).

Debugger Function Key Definitions

F1	Accesses context sensitive help. Context sensitive help is available for windows, dialog boxes, and menu items (with Ctrl+F1).
F2	Executes a single source line from the current program counter address (or a single instruction if disassembled mnemonics are mixed with source lines in the Source window).
F3	Same as F2 except when the source line contains a function call (or the assembly instruction makes a subroutine call); in these cases, the entire function (or subroutine) is executed.
F4	Break emulator execution into the monitor. You can use this to stop a running program or break into the monitor from the processor reset state.
F5	Runs the program from the current program counter address.
Shift-F4	Tiles the open debugger windows.
Shift-F5	Cascades the open debugger windows.
F7	Repeats the trace command that was entered last.
Ctrl+F7	Halts the current trace.

Starting and Exiting the Debugger

This section shows you how:

- To start the debugger
- To exit the debugger
- To create an icon for a different emulator

To start the debugger

- Double-click the debugger icon.

  Or:

1. Choose the File→Run (ALT, F, R) command in the Windows Program Manager.
2. Enter the debugger filename, C:\HP\RTC\M302\B3621.EXE (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software).
3. Choose the OK button.

   You can execute a command file when starting the debugger by using the "-C<command_file>" command line option.

To exit the debugger

1 Choose the File→Exit (ALT, F, X) command.

2 Choose the OK button.

This will end your Real-Time C Debugger session.

To create an icon for a different emulator

1 Open the "HP Real-Time C Debugger" group box, or make it active by positioning the mouse in the window and clicking the left button.

2 Choose the File→New... (ALT, F, N) command in the Windows Program Manager.

3 Select the Program Item option and choose OK.

4 In the Description text box, enter the icon description.

5 In the Command Line text box, enter the "C:\HP\RTC\M302\B3621.EXE -T<transport> -E<connectname>" command (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software). The "-T" and "-E" startup options allow you to bypass the transport and connect name definitions in the B3621.INI file.

<Transport> should be one of the supported transport options (for example, HP-ARPA, RS232C, etc.).

<Connectname> should identify the emulator for the type of transport. For example, if the HP-ARPA transport is used, <connectname> should be the hostname or IP address of the HP 64700; if the RS232C transport is used, <connectname> should be COM1, COM2, etc.

6 In the Working Directory text box, enter the directory that contains the debugger program (for example, C:\HP\RTC\M302).

7 Choose the OK button.

Working with Debugger Windows

This section shows you how:

- To open debugger windows
- To copy window contents to the list file
- To change the list file destination
- To change the debugger window fonts
- To set tab stops in the Source window
- To set colors in the Source window

To open debugger windows

- Double-click the icon for the particular window.
- Or, choose the particular window from the Window→ menu.
- Or, choose the Window→More Windows... (ALT, W, M) command, select the window to be opened from the dialog box, and choose the OK button.

To copy window contents to the list file

- From the window's control menu, choose the Copy→Windows (ALT, -, P, W) command.

  The information shown in the window is copied to the destination list file.

  You can change the name of the destination list file by choosing the Copy→Destination... (ALT, -, P, D) command from the window's control menu or by choosing the File→Copy Destination... (ALT, F, P) command.

To change the list file destination

- Choose the File→Copy Destination... (ALT, F, P) command, and select the name of the new destination list file.

- Or, from the window's control menu, choose the Copy→Destination... (ALT, -, P, D) command, and select the name of the new destination list file.

  Information copied from windows will be copied to the selected destination file until the destination list file name is changed again.

  List file names have the ".LST" extension.

To change the debugger window fonts

1. Choose the Settings→Font (ALT, S, F) command.
2. Select the font, font style, and size. Notice that the Sample box previews the selected font.
3. Choose the OK button.

To set tab stops in the Source window

1. Choose the Settings→Tabstops (ALT, S, T) command.
2. Enter the tab width. This width is also used for source lines in the trace window.
3. Choose the OK button.

The tab width must be between 1 and 20.

To set colors in the Source window

1 Exit the RTC interface and find the initialization file (B3621.INI). It should be in the directory where you installed the RTC product (C:\HP\RTC\, by default).

2 Edit the initialization file to find the "color" entry. You will see:

```
[Color]
ColorMode=ON|OFF
ColorPc=<color>
ColorSource=<color>
ColorMne=<color>
```

Where: <color> may be any of the following: RED, GREEN, BLUE, YELLOW, PINK, PURPLE, AQUA, ORANGE, SLATE, or WHITE.

- The <color> entry may be in upper-case or lower-case letters.
- When ColorMode=ON, these are the default colors:
  - ColorPC=GREEN
  - ColorSource=RED
  - ColorMne=BLUE
- The default color is black if an option is given a null value.
- The options under [Color] set colors as follows:
  - ColorPc sets the color of the line of the current program counter.
  - ColorSource sets the color of the line numbers of source lines.
  - ColorMne sets the color of the address of all mnemonic lines.

Note If you have set ColorMode=ON while using a monochrome display, you may see no line numbers in the Source window. Items that will be presented in color on a color display may not be seen at all on a monochrome display.

Using Command Files

This section shows you how:

- To create a command file
- To execute a command file
- To create buttons that execute command files

A command file is an ASCII text file containing one or more debugger commands. All the commands are written in a simple format, which makes editing easy. The debugger commands used in command files are the same as those used with break macros. For details about the format of each debugger command, refer to the "Reference" information.

To create a command file

1. Choose the File→Command Log→Log File Name... (ALT, F, C, N) command.
2. Enter the command file name.
3. Choose the File→Command Log→Logging ON (ALT, F, C, O) command.
4. Choose the commands to be stored in the command file.
5. Once the commands have been completed, choose the File→Command Log→Logging OFF (ALT, F, C, F) command.

Command files can also be created by saving the emulator configuration.

To execute a command file

1 Choose the File→Run Cmd File... (ALT, F, R) command.

2 Select the command file to be executed.

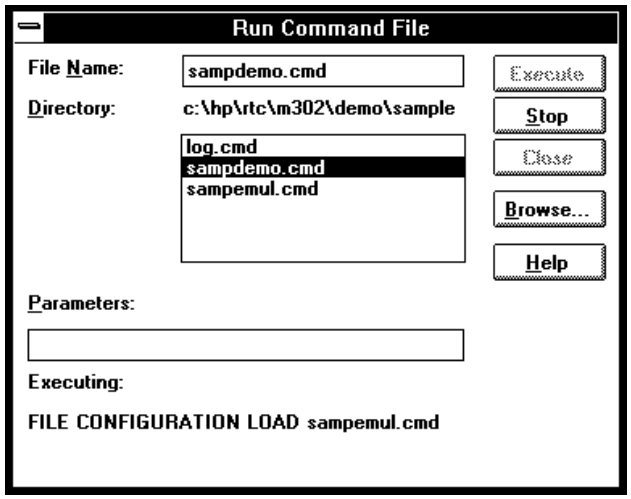
3 Choose the Execute button.

You can execute command files that have been created by logging commands.

Also, emulator configurations can be restored by executing the associated command file.

You can execute a command file when starting the debugger by using the "-C<command_file>" command line option.

Example Command File Being Executed



To create buttons that execute command files

1. Activate the Button window by clicking on the Button window icon or by choosing the Window→Button command.
2. From the Button window's control menu, choose the Edit... (ALT, -, E) command.
3. In the Command text box, enter "FILE COMMAND", a space, and the name of the command file to be executed.
4. Enter the button label in the Name text box.
5. Choose the Add button.
6. Choose the Close button.

Once a button has been added, you can click on it to run the command file.

You can also set up buttons to execute other debugger commands.

3

Configuring the Emulator

Configuring the Emulator

This chapter contains information about configuring the emulator.

- Setting the Hardware Options
- Mapping Memory
- Selecting the Type of Monitor
- Setting Up the BNC Port
- Saving and Loading Configurations
- Setting the Real-Time Options

Setting the Hardware Options

This section shows you how:

- To select the emulator clock source
- To enable or disable the target BR and BGACK signals
- To enable or disable the target BERR signal
- To enable or disable target DTACK for emulation memory accesses
- To enable or disable target system interrupts
- To enable or disable break on writes to ROM
- To specify the TRAP instruction for breakpoints
- To select the data bus width
- To specify the target memory access size
- To select internal or external cycle termination for CS0-3
- To select /IACK7 pin operation
- To specify tracing of DMA cycles
- To select the normal or dedicated interrupt mode
- To select level or edge sensitivity for /IRQ7

To select the emulator clock source

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the Internal or External option for the emulator clock source.
3. Choose the OK button to exit the Emulator Config dialog box.

When the internal clock has been selected, the emulator will operate using the internal 16 MHz oscillator.

When the external clock has been selected, the emulator will operate using the clock driven to the emulator from the target system. The maximum clock speed with the HP 64746 emulator is 16.67 MHz.

Note Changing the clock source selection resets the emulator.

To enable or disable the target BR and BGACK signals

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select or deselect the Enable target BR and BGACK signals check box.
3. Choose the OK button to exit the Emulator Config dialog box.

When the check box is selected, the /BR and /BGACK signals driven from the target system to the emulator will respond in the same manner as they would respond if the CPU were present.

When the check box is deselected, the /BR signal driven from the target system will be ignored by the emulator. The emulator will not drive an active level on /BG; the address, data, and control signals will not be placed in a tristate condition.

Internal 68302 DMA can happen regardless of the answer to this question. The /BGACK signal will always be active on internal DMA operations.

To enable or disable the target BERR signal

1 Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.

2 Select or deselect the Enable target BERR signal check box.

3 Choose the OK button to exit the Emulator Config dialog box.

When the check box is selected, the target system /BERR and the emulator /BERR signals are tied together.

When the check box is deselected, the target system /BERR and the emulator /BERR signals are disconnected. The 68302 can still generate /BERR however the target system will not see this signal. The emulator will also not respond to target system bus errors.

To enable or disable target DTACK for emulation memory accesses

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select or deselect the Emulation memory use target DTACK check box.
3. Choose the OK button to exit the Emulator Config dialog box.

The 68302 can generate an internal /DTACK. Internal DTACK can be generated by chip selects or internal processor cycles. The source of the /DTACK for chip selects in the emulator is determined by the chip select configuration questions. If the /DTACK is internally generated, this /DTACK will be driven to the target system and the emulator will not drive /DTACK to the processor. /DTACK interlock only applies to situations where the 68302 does not provide an internal /DTACK.

When the check box is deselected, emulation memory and background memory accesses are terminated by a /DTACK signal generated by the emulator.

When the check box is selected, accesses to emulation memory will not be terminated until the target system provides a /DTACK. If background cycles are being driven to the target system, the target system MUST provide a /DTACK signal to terminate background memory cycles.

When the emulator is not operating in a target system, all externally provided /DTACK cycles are terminated by an emulator generated /DTACK signal.

To enable or disable target system interrupts

1 Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.

2 Select or deselect the Enable target INT signal check box.

3 Choose the OK button to exit the Emulator Config dialog box.

When the check box is selected, the emulator detects interrupts from the target system while running in the user program or foreground monitor. Target system interrupts are ignored when the emulator is running in the background monitor.

When the check box is deselected, the emulator ignores any interrupt from the target system.

To enable or disable break on writes to ROM

1 Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.

2 Select or deselect the Enable Break on write to ROM check box.

3 Choose the OK button to exit the Emulator Config dialog box.

When the check box is selected, a running program breaks into the monitor when it writes to a location mapped as ROM.

When the check box is deselected, program writes to locations mapped as ROM do not cause breaks into the monitor.

To specify the TRAP instruction for breakpoints

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Enter a number from 0 to 0fh in the "TRAP number for Software break" text box.
3. Choose the OK button to exit the Emulator Config dialog box.

When breakpoints are set, the program opcodes are replaced with the specified TRAP instruction. The number indicates the exception vector to use in processing the TRAP.

You should select a trap number that is not used by your program code.

Note

Changing the TRAP instruction number cancels all the current breakpoints. Any breakpoints defined in target memory must be deleted by using one of the breakpoint commands.

To select the data bus width

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the 8 bits or 16 bits option for the Data Bus Width.
3. Choose the OK button to exit the Emulator Config dialog box.

The Data Bus Width selection is only used when the emulator is operating out-of-circuit.

When the emulator is operating in-circuit, the BUSW pin selects the data bus width.

To specify the target memory access size

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the 8 bits or 16 bits option for the Target Memory Access Size.
3. Choose the OK button to exit the Emulator Config dialog box.

You can only specify the target memory access size when the 16 bit data bus width is selected.

All target system memory accesses occur in the specified size.

To select internal or external cycle termination for CS0-3

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the Internal or External option for chip selects CS0, CS1, CS2, and/or CS3.
3. Choose the OK button to exit the Emulator Config dialog box.

If internal is selected, an active level on /CS[0-3] causes the processor /DTACK to be driven to the target system. The emulator does not drive /DTACK to the processor.

If external is selected, an active level on /CS[0-3] causes the emulator to drive /DTACK to the processor, from either the target system or the emulator based on the Emulation Memory Use Target DTACK setting.

To select /IACK7 pin operation

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the /IACK7 or PB0 option for the /IACK7 Pin.
3. Choose the OK button to exit the Emulator Config dialog box.

If /IACK7 is selected, the /IACK7-PB0 pin is used as /IACK7. Emulator level 7 interrupt acknowledges are blocked to the target.

When PB0 is selected, the emulator does not affect this line.

To specify tracing of DMA cycles

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the disable, enable, or tag option for Arbitration Analysis.
3. Choose the OK button to exit the Emulator Config dialog box.

When the tag arbitration analysis mode is selected, a single emulation analyzer state will be generated each time an external bus arbitration sequence occurs.

When arbitration analysis is disabled, no emulation analyzer states are created to mark the DMA transaction.

When arbitration analysis is enabled, the emulation analyzer will look at the processor strobes to generate analyzer states showing the DMA address, data and status information.

To select the normal or dedicated interrupt mode

1 Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.

2 Select either the normal or dedicated option for the Interrupt Mode.

3 Choose the OK button to exit the Emulator Config dialog box.

This options tells the emulator which interrupt mode the 68302 is programmed for -- it does not actually program the 68302 interrupt mode. If the 68302 is not in the selected mode when a break is attempted, the break will either fail or lead to unexpected interrupts.

When the interrupt mode is normal, the interrupts are encoded on /IPL0, /IPL1, and /IPL2.

When dedicated, /IPL2 becomes /IRQ7, /IPL1 becomes /IRQ6, and /IPL0 becomes /IRQ1. In dedicated mode, edge or level sensitivity is selected via the /IRQ7 mode checkbox.

To select level or edge sensitivity for /IRQ7

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select either the level or edge option for the /IRQ7 mode.
3. Choose the OK button to exit the Emulator Config dialog box.

You can only select level or edge sensitivity for /IRQ7 when the dedicated interrupt mode is selected.

When the /IRQ7 mode is level, the interrupt 7 will happen on a low level on /IRQ7.

When the mode is edge, a falling edge on /IRQ7 causes an interrupt.

Mapping Memory

This section shows you how:

- To map memory

To map memory

1. Choose the Settings→Emulator Config→Memory Map... (ALT, S, E, M) command.
2. Specify the starting address in the Start text box.
3. Specify the end address in the End text box.
4. If necessary, select the function code from the Function Code drop-down list.
5. Select the memory type in the Type option box.
6. Choose the Apply button.
7. Repeat steps 2 through 6 for each range to be mapped.
8. Choose the Close button to exit the Memory Map dialog box.

You can map up to 7 address ranges (map terms). The minimum amount of emulation memory that can be allocated to a range varies between 512 and 1024 bytes, depending on the capacity of the memory board used.

It's only necessary to specify *function codes* when mapping overlapping address ranges for different memory spaces. When mapping overlapping ranges, you can only select function codes that haven't already been selected for previously mapped ranges.

You can specify one of the following memory types for each map term:

eram	Specifies "emulation RAM".
erom	Specifies "emulation ROM".
tram	Specifies "target RAM".
trom	Specifies "target ROM".
guarded	Specifies "guarded memory".

When breaks on writes to ROM are enabled in the emulator configuration, any access from the user program to any memory area mapped as ROM stops the emulator.

For non-mapped memory areas, select any of the memory types in the Default option box.

Note that the internal memory space of the 68302 must be mapped as target RAM. The BAR and SCR registers (located at 0F0H and 0F4H in the exception vector table) may be mapped as emulation RAM, but you should use the register commands to modify or examine these locations.

You should map all memory ranges used by your programs before loading programs into memory.

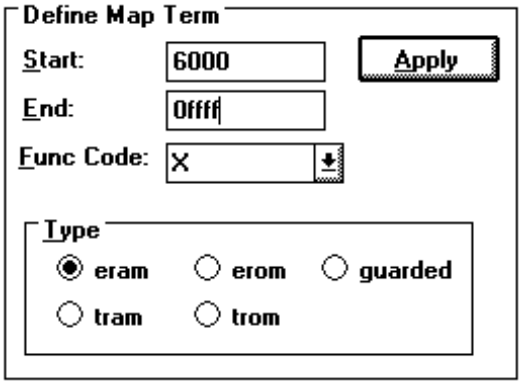
Note

The monitor requires 2 Kbytes of emulation memory.

To delete a map term, first select it in the Map list box; then, choose the Delete button.

Example

To map addresses 6000h through 0fffffh as an emulation RAM having "X" function code, specify the mapping term as shown below.



Choose the Apply button to register the current map term.

Then, choose the Close button to quit mapping.

Selecting the Type of Monitor

This section shows you how:

- To select the background monitor
- To select the foreground monitor

Refer to "Monitor Program Options" in the "Concepts" part for a description of emulation monitors and the advantages and disadvantages of using background or foreground emulation monitors.

Note

Select the type of monitor before mapping memory because changing the monitor type resets the memory map.

To select the background monitor

1. Choose the Settings→Emulator Config→Monitor... (ALT, S, E, O) command.
2. Select the Background option.
3. Specify whether background monitor cycles should be driven to the target system by either selecting or deselecting the Drive Monitor Cycles to Target check box.
4. Choose the OK button.

If target system circuitry (for example, an external watchdog timer) depends on the constant appearance of cycles on the microprocessor bus, you can drive background monitor cycles to the target system.

If background cycles are to be driven to the target system:

- Specify whether these cycles should be terminated by target system /DTACK signals.
- Specify the value to be output on the upper address lines.
- Specify the function code to be used for driven background cycles.

When you select background cycles to be terminated by target system /DTACK signals, /BERR signals will also terminate a monitor cycle and cause the emulator to begin execution of the bus error handler.

To select the foreground monitor

1. Edit the foreground monitor program source file to define its base address.
2. Assemble and link the foreground monitor.
3. Choose the Settings→Emulator Config→Monitor... (ALT, S, E, O) command.
4. Select the Foreground option.
5. Enter the stack address in the Reset Value for Supervisor Stack text box. The foreground monitor requires a stack to be set up in the user program. The stack address must be an even address in a RAM area.
6. Enter the base address of the foreground monitor in the Monitor Address text box. This is the same address that was defined in step 1.
7. Enter the name of the foreground monitor object file in the Monitor Name text box.
8. If you wish to designate the foreground monitor memory space as supervisor space, select the Monitor Located at Supervisor FC check box. Otherwise, no function code is specified.
9. Choose the OK button.
10. Use the Settings→Emulator Config→Memory Map... (ALT, S, E, M) to remap the user program memory areas. Selecting the foreground monitor automatically resets the current memory map.

11 Load the foreground monitor by choosing the Execution→Reset (ALT, E, E) command or by choosing the File→Load Object... (ALT, F, L) command and entering the name of the foreground monitor object file.

12 Load the user program by choosing the File→Load Object... (ALT, F, L) command and entering the name of the user program object file.

13 Modify the TRACE vector (24H) in the processor's exception vector table to point to the TRACE_ENTRY symbol in the monitor program. This is required by the single-step feature.

When the monitor needs to respond to target system interrupts or must be modified in some other way, you can use a foreground monitor program.

For more information on the foreground monitor, refer to the "Monitor Program Options" section in the "Concepts" information.

Note

Selecting the foreground monitor automatically resets the memory map and maps a new term for the foreground monitor.

Setting Up the BNC Port

This section shows you how:

- To output the trigger signal on the BNC port
- To receive an arm condition input on the BNC port

To output the trigger signal on the BNC port

- **Choose the Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O) command.**

  The HP 64700 Series emulators have a BNC port for connection with external devices such as logic analyzers or oscilloscopes.

  This command enables the trigger signal from the internal analyzer to be fed to external devices.

To receive an arm condition input on the BNC port

- **Choose the Settings→BNC→Input to Analyzer Arm (ALT, S, B, R) command.**

  The HP 64700 Series emulators have a BNC port for connection with external devices such as logic analyzers or oscilloscopes.

  This command allows an external trigger signal to be used as an arm (enable) condition for the internal analyzer.

Saving and Loading Configurations

This section shows you how:

- To save the current emulator configuration
- To load an emulator configuration

To save the current emulator configuration

1. Choose the File→Save Emulator Config... (ALT, F, V) command.
2. In the file selection dialog box, enter the name of the file to which the emulator configuration will be saved.
3. Choose the OK button.

This command saves the current hardware, memory map, and monitor settings to a command file.

Saved emulator configuration files can be loaded later by choosing the File→Load Emulator Config... (ALT, F, E) command or by choosing the File→Run Cmd File... (ALT, F, R) command.

See Also

File→Save Emulator Config... (ALT, F, V) in the "Menu Bar Commands" section of the "Reference" information.

To load an emulator configuration

1. Choose the File→Load Emulator Config... (ALT, F, E) command.
2. Select the name of the emulator configuration command file to load from the file selection dialog box.
3. Choose the OK button.

This command lets you reload emulator configurations that have previously been saved.

Emulator configurations consist of hardware, memory map, and monitor settings.

Setting the Real-Time Options

This section shows you how:

- To allow or deny monitor intrusion
- To turn polling ON or OFF

The monitor program is executed by the emulation microprocessor when target system memory, memory-mapped I/O, and microprocessor registers are displayed or edited. Also, periodic polling to update the Memory, I/O, WatchPoint, and Register windows can cause monitor program execution.

This means that when the user program is running and monitor intrusion is allowed, the user program must be temporarily interrupted in order to display or edit target system memory, display or edit registers, or update window contents.

If it's important that your program execute without these kinds of interruptions, you should deny monitor intrusion. You can still display and edit target system memory and microprocessor registers, but you must specifically break emulator execution from the user program into the monitor first.

When monitor intrusion is denied, polling to update window contents is automatically turned OFF.

When monitor intrusion is allowed, you can turn polling for particular windows OFF to lessen the number of interruptions during user program execution.

To allow or deny monitor intrusion

- To deny monitor intrusion, choose the RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command.

- To allow monitor intrusion, choose the RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command.

When you deny monitor intrusion, any debugger command that may interrupt a running user program is prevented. This ensures the user program execute in real-time.

When you allow monitor intrusion, debugger commands that may temporarily interrupt user program execution are allowed.

The current setting is shown by a check mark (√) next to the command.

To turn polling ON or OFF

- To turn I/O window polling ON or OFF, choose the RealTime→I/O Polling→ON (ALT, R, I, O) or RealTime→I/O Polling→OFF (ALT, R, I, F) command.

- To turn WatchPoint window polling ON or OFF, choose the RealTime→Watchpoint Polling→ON (ALT, R, W, O) or RealTime→Watchpoint Polling→OFF (ALT, R, W, F) command.

- To turn Memory window polling ON or OFF, choose the RealTime→Memory Polling→ON (ALT, R, M, O) or RealTime→Memory Polling→OFF (ALT, R, M, F) command.

  When the user program is running and monitor intrusion is denied, polling is automatically turned OFF.

  When the user program is running and monitor intrusion is allowed, you can turn polling OFF to reduce the number of user program interrupts made in order to update I/O, WatchPoint, and Memory window contents.

  The current settings are shown by check marks (√) next to the command.

4

Plugging the Emulator into Target Systems

Plugging the Emulator into Target Systems

This chapter contains information about plugging the emulator into target systems and configuring the emulator so that it operates correctly.

- Connecting the Emulator Probe
- Configuring the Emulator for In-Circuit Operation

Connecting the Emulator Probe

This section shows you how:

- To plug-in the emulator probe

To plug-in the emulator probe

To avoid having to replace the entire probe because of a bent or broken pin, use a pin protector (that is, an extra PGA socket) between the probe and the target.

PGA sockets are available from HP as HP part number 1200-1318. A MacKenzie Technology PGA-100M-003B1-1324 socket should also be suitable.

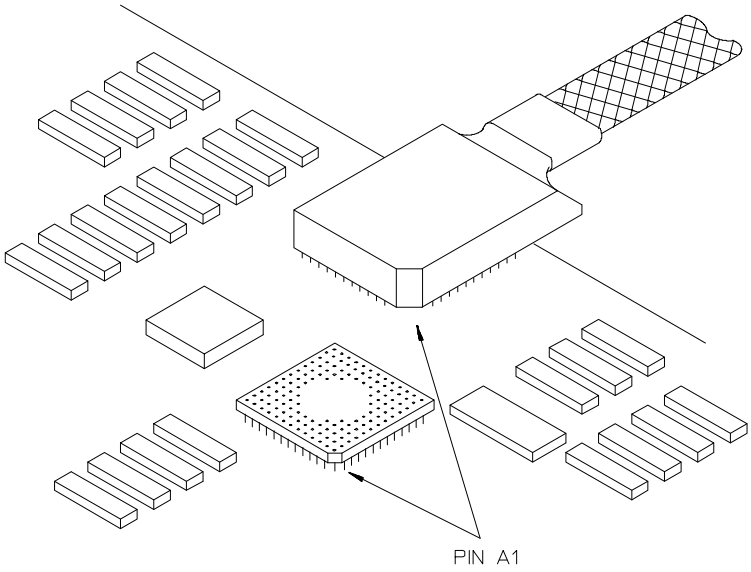
To prevent emulator and probe components from being damaged by static electricity, store and use the emulator in a place resistant to static electricity.

1 Turn OFF power to the target system.

2 Turn OFF power to the emulator.

3 Remove the processor from the target system.

4 Connect the probe to the target system



CAUTION: MAKE SURE PIN A1 OF THE PROBE CONNECTOR IS ALIGNED WITH PIN A1 OF THE SOCKET. DAMAGE TO THE EMULATOR PROBE WILL RESULT IF THE PROBE IS INCORRECTLY INSTALLED!

5 Be sure to orient the probe correctly. Pin A1 of the PGA matrix is at the notched corner of the probe. (Note that pin "A1" of the PGA matrix is signal "A14". Pin numbers DO NOT correspond to signal numbers for the 68302.)

6 Turn ON power to the emulator.

7 Turn ON power to the target system.

8 Turning ON power to the emulator before turning ON power to the target system will prevent damage to sensitive components in the target system.

9 Start the debugger.

Configuring the Emulator for In-Circuit Operation

Many users of the 68302 emulator encounter problems when first plugging the emulator into their target system. This section should help you avoid or quickly correct most of those problems.

- Step 1. Understand the important concepts
- Step 2. Set up your chip-selects
- Step 3. Reprogram chip-select base addresses
- Step 4. Know your interrupt mode
- Step 5. Decide whether to use the foreground monitor
- Step 6. Set up the emulator for the foreground monitor
- Step 7. If you use the 68302 built-in DRAM refresh
- Step 8. Set up the DTACK signals
- Step 9. If emulator status shows HALTED
- Step 10. Choose the correct target memory access size
- Step 11. Check your DTACK pullup resistor!
- If you have problems

Step 1. Understand the important concepts

There are a few basic concepts related to 68302 emulation that should be understood before you begin. Understanding these concepts will help you avoid the common startup problems.

Accessing Memory that is Enables by a Chip-Select

Nearly all 68302 target systems rely on the built-in chip-selects for at least some accesses to memory. The important concept to remember is that the 68302 emulator does not automatically setup any chip-selects for you other than chip-select zero, which is automatically setup by the 68302 itself.

If you attempt to access memory that is dependent on a chip-select being programmed, without first ensuring that that chip-select is programmed, some type of failure will result. This most often causes problems when you are trying to do commands such as load memory, display memory, run or step.

Setting the Interrupt Mode to Dedicated or Normal

The 68302 has two basic types of external interrupts, referred to as Normal or Dedicated mode. In Normal mode, three lines are used to indicate interrupt levels 0 through 7 or no interrupt. This normal mode scheme is just like the one used on traditional 680X0 devices. In Dedicated mode, the three lines each have a dedicated purpose, one for each of the interrupts level 1, 6, and 7.

Why is this important? The emulator uses a level 7 interrupt to accomplish what is known as a "break" from the user program to the emulation monitor program. The monitor is needed for tasks such as display/modify of target memory and 68302 registers, as well as single-stepping. In order to initiate a level 7 interrupt, the emulator must know what interrupt mode the 68302 has been programmed for. An emulator configuration option is used to indicate which mode the 68302 will be in. If the 68302 is not in that mode when a break is attempted, the break will either fail or lead to unexpected interrupts.

Note that although the 68302 emulator uses the level 7 interrupt, you are free to use the level 7 interrupt for your own purposes. This is because the emulator is able to differentiate between a target system-generated interrupt and and an emulator-generated interrupt.

The Freeze Pin and the Background Monitor

The 68302 has a hardware input pin called FRZ which can be used to "freeze" selected on-chip peripherals. When configured for the background monitor, the 68302 emulator will assert the FRZ pin during the entire time the emulator is running in the monitor.

If you are using the SCC functions of the 68302 or the built-in DRAM refresh feature, you will most likely want to configure your emulator for use with the

foreground monitor since these functions will be "frozen" during all background monitor operation. "Step 5. Decide whether to use the foreground monitor" will further explain these issues.

While running in the background monitor, the watchdog timer will not be serviced. However, knowing that the FRZ pin is asserted during background monitor operation makes it easy for you to freeze the watchdog timer during this time. Freezing the watchdog timer will be discussed later.

On-Chip Locations

Most of the special features of the 68302 are controlled via a 4 Kbyte block of on-chip locations (RAM and special registers). The address of that 4 Kbyte block is determined by the value written to the BAR. When you configure the emulator, you must map that 4 Kbyte block of memory as target RAM. Mapping that block as "guarded" or "emulation" memory will prevent proper operation or result in guarded memory access errors.

Any display or modify of on-chip locations, including registers, requires the emulation monitor program. If you do a display or modify of these locations or registers while running your program, the emulator will briefly break into the monitor, perform the display or modify, then return to your program.

Step 2. Set up your chip-selects

Failure to setup the chip-select registers is by far the most common cause of problems when using the 68302 emulator. If you remember that load, modify/display, step and run commands often rely on valid chip-select settings, you can avoid most of the common mistakes made by users.

The 68302 has 4 chip-selects, only one of which is enabled after a reset condition. Nearly all 68302 target systems rely on at least one chip-select for accesses to memory. If you are going to access any target memory that relies on a chip-select, then you MUST be sure that the appropriate chip-select registers are initialized first. In some cases, even executing code from emulation memory will require that you first initialize your chip selects. This can be done in one of two ways:

Method 1: Using a Command File to Setup Chip-Selects

Use a series of commands or a command file that modifies the registers to the values you will need. For example, here is a command file that sets up CS0 and CS1.

```
RESET
BREAK
REG bar TO 800h
REG or0 TO 3fc2h
REG br0 TO 0001h
REG or1 TO 1f80h
REG br1 TO 0801h
```

Note

It is important that you first modify the BAR register BEFORE you attempt to modify the chip-select registers because the location of those registers is calculated based on the value in the BAR.

The above example will map the Internal system registers to location 800XXXH, and then initialize CS0 as a read-only, 1 wait-state block from 0 through 01FFFFH and CS1 as a read-write, 0 wait-state block from 400000H through 43FFFFH.

If you are going to load your initialization code into target RAM using the emulator, then you must use Method 1.

Method 2: Using Your Initialization Code to Setup Chip-Selects

Execute a small section of your initialization code that sets up the proper values in the chip select registers, and break into the monitor immediately after that, using either a software or analysis breakpoint.

For example, assuming you have the area from 0 through 01FFFFH mapped as emulation ROM and have already written your code to initialize the chip-select registers, you should load that code, set a breakpoint, and then run from a reset condition. When the breakpoint is hit, the chip-selects will have been properly initialized. Here is an example program that will demonstrate this:

```
XDEF     CS_INIT
ORG      $0
DC.L     $440000          ; Stack begins at $43FFFE
DC.L     $400             ; Reset initialization code

ORG      $400
```

```
            MOVE.W   #$0800,$F2        ; Set up the BAR for $800XXX
            MOVEA.L  #$800000,A0
            MOVE.W   #$3F02,($832,A0)  ; OR0 - 512K, read-only, 1 wait-state
            MOVE.W   #$0001,($830,A0)  ; BR0 - base address 0
            MOVE.W   #$1F80,($836,A0)  ; OR1 - 256K, read-write, 0 wait-state
            MOVE.W   #$0801,($834,A0)  ; BR1 - base address $400000
            NOP
CS_INIT     NOP
            NOP
```

Assuming you have loaded the above example into emulation memory, you can now set a breakpoint on CS_INIT and run from reset.

If your initialization code is loaded in target ROM, you will not be able to set a software breakpoint there. In this case, you should use an analysis breakpoint by tracing about CS_INIT, selecting the break on trigger option, and running from reset.

Note

When using the trace command with the "break on trigger" option, be sure to select an address that is at least 2 words after the instruction you expect to be executed. This ensures that the analyzer does not break on a "prefetch" of that instruction.

When the emulator breaks into the monitor the chip-select registers will have been initialized to the values you're using. You can now use display, modify, load, step or run commands because the chip-selects are properly initialized.

Step 3. Reprogram chip-select base addresses

If you are not going to be changing the base address of chip-select 0 from address 0, or changing the base address registers of the other chip-selects after initial setup, you should skip this step. Dealing with chip-selects whose base addresses are being changed is probably the most difficult challenge you will face with the 68302 emulator.

Before you can successfully emulate in this mode, it is important that you understand how the 68302 uses chip-selects. When you enter memory map terms in your configuration, you will notice that you enter all address ranges without the option to qualify a specific range with a chip-select number. What this means is that the emulation memory mapper cannot "track" changes made to the base address of any chip-selects. Any emulation

memory ranges you map respond based on the value on the address bus (and optionally the function code). The chip-select signals are not used by the emulator to decode memory map terms.

What does this mean? This means that if you are going to be mapping a block of memory to either emulation RAM or emulation ROM and are going to be changing the base address of that block, you must ensure that valid code or data exists at BOTH blocks of memory. For example, if you have boot code that originally exists at address 0 and chip-select 0 is later reprogrammed with a base address of 400000H, you must ensure that the code exists at both 0 and at 400000H. If you are executing your code from target memory, this is easy, since your target hardware will be designed to decode memory based on chip-select 0 being active, not based on the full address (A23-A1).

Here is an example of how you can emulate a setup where the chip-select base addresses are being changed. The technique of switching the RAM and ROM addresses closely follows an example that Motorola provides in Revision 2 of the MC68302 User's Manual.

We start with the following memory map:

```
000000..00FFFF@x eram
400000..41FFFF@x eram
800000..800FFF@x tram
```

and use the following program:

```
            XDEF     SWITCHED
            ORG      $400000
            DC.L     $10000             ; Stack begins at $0FFFE
            DC.L     RESET-$400000      ; Reset initialization code
            DC.L     BUS_ERROR
            DC.L     ADDR_ERROR

            ORG      $400400
RESET       MOVE.W   #$0800,$F2         ; Set up the BAR for $800XXX
            MOVEA.L  #$800000,A0
            MOVE.W   #$3FC2,($832,A0)   ; OR0 - 128K, read-only, 1 wait-state
            MOVE.W   #$0001,($830,A0)   ; BR0 - base address 0
            MOVE.W   #$1FE0,($836,A0)   ; OR1 - 64K, read-write, 0 wait-state
            MOVE.W   #$0801,($834,A0)   ; BR1 - base address $400000
            NOP
CP_VECTORS  MOVEA.L  #0,A3              ; pointer to vector table
            MOVEA.L  #$400000,A4        ; where to copy it
DO_COPY     MOVE.L   (A3)+,(A4)+
            CMPA.L   #$400,A3            ; at the end?
            BNE      DO_COPY
            NOP                         ; now the vector table is in ROM and RAM
            NOP
CS_INIT     NOP

; the following instructions move code into on-chip RAM, then jump to it.
```

```
; A2 is used to point to where ROM WILL be.

            MOVEA.L  #SWITCHED,A2
            MOVEA.L  A0,A1
            MOVE.L   #$317CA001,(A1)+
            MOVE.L   #$0834317C,(A1)+
            MOVE.L   #$C8010830,(A1)+
            MOVE.W   #$4ED2,(A1)+
            JMP      (A0)
            NOP
SWITCHED    NOP                        ; it's switched now!!!!!!!
            NOP
            MOVE.L   0,D0              ; read vector 0
            NOP

BUS_ERROR   JMP      BUS_ERROR
ADDR_ERROR  JMP      ADDR_ERROR
```

What happens first? The first thing that needs to be done is for you to copy your startup code that will exist in ROM beginning at 400400H to address 400H because what will eventually be address 400400H is address 400H immediately following a reset (because chip-select 0 maps to address 0). To do this, choose the Utilities→Copy... (ALT, -, U, C) command from the Memory window's control menu, enter 400000h as the start address, enter 401fffh as the end address, and enter 0 as the destination address.

This will cause the first 8 Kbytes of code that was loaded at 400000H to be copied to emulation RAM beginning at 0. Doing this will make the emulator act like your target system will at reset. Now you can simply set a breakpoint on SWITCHED and run from reset.

When the emulator breaks into the monitor, chip-select 0 will have been programmed for address range 400000H through 41FFFFH, and chip-select 1 will have been programmed for address range 0 through 0FFFFH. Your program can now modify RAM at address 0.

Note that an important limitation of this method is that if a target system reset occurs while your program is running, the program will fail. This is because after a reset, chip-select 0 will again map to address 0, but your code is no longer at address 0. In order to put your code at 0, you would again need to choose the Utilities→Copy... (ALT, -, U, C) command from the Memory window's control menu, enter 400000h as the start address, enter 401fffh as the end address, and enter 0 as the destination address.

There are other ways that you can debug a system that reprograms the base addresses of chip-selects. Another approach would be to debug your system in two separate steps. First, you can debug your initialization code. Once your initialization code is debugged, you can configure the emulator and map memory based on what your final chip-select addresses will be. You can then

use a command file to set the chip-selects to their final addresses. This may require some vector table changes.

Still another approach is to use only target memory in areas where chip-selects are used. This doesn't require any special steps since your target memory will "track" the chip-select changes.

Step 4. Know your interrupt mode

If your target does not use dedicated mode interrupts, you can skip this step.

One of the more common problems users encounter is not being able to break into the monitor (ERROR: break failed). A common cause of this is selecting the dedicated interrupt mode in the emulator hardware configuration, but failing to ensure that the 68302 is programmed for dedicated mode interrupts.

The emulator is configured to use normal mode interrupts by default. If you will be setting up the 68302 for dedicated mode interrupts, choose the Settings→Emulator Config→ Hardware... (ALT, S, E, H) command and select the dedicated Interrupt Mode option.

The interrupt mode of the 68302 is determined by the value programmed in the GIMR register. If you are using dedicated mode interrupts, you must set the most significant bit of the GIMR register BEFORE you attempt to step or break. Similar to setting up chip-selects, you have two options for making sure you have a valid interrupt mode:

Method 1: Using a Command File to Set the Interrupt Mode

Use a series of commands or a command file that modifies the GIMR to force the 68302 to be in dedicated mode. For example, here is a command file that sets up the GIMR for dedicated mode:

```
RESET
BREAK
REG bar TO 800h
REG gimr TO 8700h
```

Note

It is important that you first modify the BAR register BEFORE attempting to modify the GIMR register because the location of that register is calculated based on the value in the BAR.

Note that doing a break when the 68302 is in a reset state does not require a level 7 interrupt, and therefore will work regardless of the interrupt mode setting.

Method 2: Using Your Initialization Code to Set the Interrupt Mode

Execute a small section of initialization code that sets the GIMR register to the proper value, and break into the monitor immediately after that using either a software or analysis breakpoint.

Here is an example that shows how this works using an analysis breakpoint. In this case, an analysis breakpoint is useful because it will confirm that everything is working properly (if you use a software breakpoint the emulator does not issue a level 7 interrupt, so that would not test the interrupt mode):

Start with the following program loaded in either emulation or target memory:

```
            XDEF     CS_INIT,GIMR_INIT
            ORG      $0
            DC.L     $440000           ; Stack begins at $43FFFE
            DC.L     $400              ; Reset initialization code

            ORG      $400
            MOVE.W   #$0800,$F2        ; Set up the BAR for $800XXX
            MOVEA.L  #$800000,A0
            MOVE.W   #$3F02,($832,A0)  ; OR0 - 512K, read-only, 1 wait-state
            MOVE.W   #$0001,($830,A0)  ; BR0 - base address 0
            MOVE.W   #$1F80,($836,A0)  ; OR1 - 256K, read-write, 0 wait-state
            MOVE.W   #$0801,($834,A0)  ; BR1 - base address $400000
            NOP
CS_INIT     NOP
            NOP
            MOVE.W   #$8740,($812,A0)  ; set for dedicated mode interrupt
            NOP
            NOP
GIMR_INIT   NOP
            NOP
            NOP
            NOP
```

Assuming you have loaded the above example into emulation or target memory, you can now trace about GIMR_INIT, selecting the break on trigger option, and run from reset.

When the emulator breaks into the monitor the 68302 will be in dedicated mode. You can now use the step and break command.

Note that if you are just starting to debug your initialization code, and would like to step through each of your startup instructions, you should use Method 1.

Step 5. Decide whether to use the foreground monitor

The default configuration for the 68302 emulator uses the background monitor. The background monitor is easier to use and usually requires no special setup.

The foreground monitor offers you much more flexibility but also requires extra setup and sometimes requires more in-depth knowledge of your target system. You should use the foreground monitor if any of the following are true:

- Your target system has DRAM and you use the built-in DRAM refresh of the 68302 to refresh it.
- You need to service some type of interrupt even when you are running in the monitor. An example of this might be a timer interrupt that MUST be serviced on a regular interval.
- You need to have SCC functions occurring while you are running in the monitor.
- You need to customize the monitor to provide some special function, such as servicing an off-chip watchdog timer.
- You have frequent external interrupts occurring and this prevents you from being able to single-step correctly. This problem can occur when external interrupts are left pending during background execution. The problem will appear as a PC that is "stuck" at the same address after each step command. (One solution to this problem is to modify the processor status register to raise the interrupt mask before stepping, and then restoring it after you are finished stepping.)

Note that some users choose the foreground monitor so that they can customize it to refresh the on-chip watchdog timer of the 68302; however,

this is not necessary. The 68302 SCR register contains a bit, which when set will suspend the watchdog timer whenever the FRZ pin is asserted. This bit is called "FRZW". You can simply modify your code to set that bit on startup and the watchdog timer will automatically be suspended when you are in the background monitor.

If you do choose to use the foreground monitor, you cannot take advantage of the FRZW feature since the FRZ pin is only asserted in the background monitor. Note that the FRZ1 and FRZ2 bits of the SCR can likewise be used to suspend timer1 and timer2 during background operation.

Step 6. Set up the emulator for the foreground monitor

Skip this step if you have decided not to use the foreground monitor.

Here are the steps you should follow if you choose to use the foreground monitor:

1 Choose an address for the monitor. Unlike the background monitor, the foreground monitor resides in the same memory space as your program (hence the term "foreground"). You must first select the address where the monitor will reside. Here are some considerations:

- The monitor MUST reside in emulation memory. Note that when you configure the emulator and answer the questions regarding the foreground monitor the memory map will automatically be modified to include a block of emulation RAM for the monitor.

- All monitor cycles (both reads and writes) WILL be seen by your target system, therefore, you must choose an address that your target system will "tolerate". For example, if your target has a range of memory that is decoded for read-only accesses you should not locate the monitor there.

- You should not locate the monitor in area that your program may overwrite. The monitor must be mapped to emulation RAM, so you must be careful to not allow it to get overwritten.

Note If you are using 1 Mbyte SIMMs in your 68302 emulator, you MUST locate the monitor on an 8 Kbyte boundary; otherwise, locate the the monitor on a 2 Kbyte boundary.

Note A defect in firmware revision A.00.03 of the 68302 emulator will prevent the monitor from successfully loading at any address whose first non-zero digit is not numeric. For example, you would not be able to locate the monitor at 0A0000H, 0B0000H, 0A000H, etc. You could load it at address 1A0000H, 3B0000H, etc. To check the revision of your 68302 firmware issue the command "ver" from the terminal interface This problem does not exist in firmware revision A.00.04.

2 Make a local copy of the monitor source file, C:\HP\RTC\M302\FGMON\FGMON.S (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software).

3 Modify the monitor source file. You need to uncomment the "ORG" directive and put the address chosen in step number 1 in the address field.

4 Assemble and link the monitor.

For the HP AxLS assembler, use the command:

```
as68k -L fgmon.s > fgmon.lst
```

For the Microtec 68000 assembler, use the command:

```
asm68k -l fgmon.s > fgmon.lst
```

5 Link the monitor.

For the HP AxLS linker, use the command:

```
ld68k -o fgmon fgmon.o
```

For the Microtec linker, use the command:

```
lnk68k -o fgmon fgmon.o
```

6 Modify your emulator configuration to specify a foreground monitor, specify the address you chose and the filename you created (in this example, the filename is FGMON.X for the Microtec or HP AxLS linker).

7 Make sure that the address range 0FFF000H through 0FFFFFFH is mapped as "target RAM". This is important because the emulator will attempt to access the on-chip locations of the 68302 upon entering the monitor. It uses the default address for the BAR, placing the internal register section at address 0FFF000H. Leaving this as guarded will cause a guarded access error, mapping it to emulation RAM will cause unpredictable results.

If You Plan to Single-Step

If you plan to single-step when using the foreground monitor there's one more thing you need to do. You need to ensure that the vector table entry at location 24H points to the label "TRACE_ENTRY" in the foreground monitor module. You can determine what this address is by looking at the listing file created when you assembled the monitor. There are many approaches you can take to ensure this, including:

- Modify your own code to place the value of TRACE_ENTRY at location 24H. This is probably the most robust method because you can account for things such as a RAM-based vector table that is loaded by your initialization code.
- Use a modify memory command to put the value of TRACE_ENTRY at 24H after loading your program. This works fine if your program doesn't overwrite or re-initialize the vector table during execution.
- Use a command file to single step that first modifies memory location 24H to point to TRACE_ENTRY, then steps. For example:

```
MEMORY FILL LONG 24..27 XXXXXXH
STEP
```

  where XXXXXXH is the address of TRACE_ENTRY

  You shouldn't need to use this method unless your program or operating system is continuously changing the vector table contents.

If you are mapping the vector table to a target ROM area and are not able to modify location 24H, you can map the vector table area to emulation RAM, copy the contents of your target ROM to emulation RAM, then modify location 24H. Here are the steps needed to do this:

1. Map 0 through 3FFH as emulation RAM.
2. From the Memory window's control menu, choose the Utilities→Image... (ALT, -, U, I) command. Enter a start address of 0, an end address of 03ffh, and choose the OK button.
3. Modify the long word of memory at address 24H to the address of TRACE_ENTRY.

Note

The Utilities→Image... (ALT, -, U, I) command allows you map a section of memory as emulation RAM or ROM and copy the contents of your target memory to that section. In this example, it is used to read all locations 0 through 3FFH from your target ROM, then write those values to emulation memory. This is a convenient way of making changes to your target ROM without having to burn new ROMs.

If You Need Interrupts Serviced While Running in the Monitor

By default, the foreground monitor leaves the Interrupt Mask of the 68302 at 7 after a "break" into the monitor. If you wish to have any interrupts other than level 7 interrupts serviced while the monitor is running, you must modify the monitor source code. The monitor source has the following instructions commented out:

```
;  MOVE.W       PSTAT,D0
;  OR           #0F8FFH,D0
;  MOVE.W       SR,D1
;  AND          D1,D0
;  MOVE.W       D0,SR
```

Simply uncomment the instructions shown above and reassemble and relink the monitor; then, reload your configuration.

Note that the monitor is not reentrant. This means that you should be careful not to cause a "break" into the monitor at a time when your program may have interrupted the monitor. An example of this would be a case where you set a breakpoint inside one of your interrupt service routines, and that interrupt service routine gets called as a result of an interrupt to the monitor program.

Step 7. If you use the 68302 built-in DRAM refresh

The 68302 has a special built-in function that allows for automatic refresh of DRAM with no additional target hardware. If your target system does not rely on this built-in feature of the 68302 you can skip this step. If you do use the built-in DRAM refresh feature, you must use the foreground monitor; otherwise, DRAM values will become corrupted when you break into the background monitor.

In order to make the DRAM refresh work, you should follow the steps outlined in Step 6. Set up the emulator for the foreground monitor. You will need to make one addition to the foreground monitor source code to ensure that DRAM refreshes occur properly while in the monitor. To make the addition, edit the monitor source and find the following section:

```
TARG_MEM_WR
   MOVE.L       TARG_START,A0
   MOVE.L       #XFR_BUF,A1
   MOVE.W       TARG_BYTES,A2
   CMP.W        #00800H,A2
   BPL          TARG_WORD_WR
```

Then add the next two lines immediately after the BPL instruction:

```
CMPA.L       #$F2,A0      ; If write to 0F2H then force a word write
BEQ          TARG_WORD_WR
```

Why is this needed? Whenever a break into the monitor occurs, the emulator always checks the value in the BAR and rewrites it. Without this addition, the emulator will make two separate byte accesses to locations 0F2H and 0F3H. If a DRAM refresh occurs in between the write to 0F2H and 0F3H, the DRAM refresh operation will not function properly. Making this change ensures that a DRAM refresh will not occur during a write to the BAR.

After making this change, you should reassemble and relink the monitor; then, reload your configuration.

Step 8. Set up the DTACK signals

Probably the least understood configuration options relate to the interlocking and source for the DTACK signal. Selecting the options correctly is easy once you know a little bit about your target system.

Selecting the DTACK sources for chip-selects CS0 through CS3 is easy. If you will be programming a chip-select to generate DTACK internally (as is most often the case) you should select "internal"; otherwise, select "external". You may notice that the default for CS0 is "internal" and all others "external". This is because the 68302 CS0 is configured to generate an internal DTACK by default.

A somewhat more difficult option is "Emulation Memory Use Target DTACK". Selecting this option will force the emulator to wait for your target system to assert the DTACK signal whenever an access occurs to emulation memory AND there is no internally generated DTACK for this cycle.

The "Emulation Memory Use Target DTACK" option is not selected by default and can most often be left that way. When do you need to answer yes? If ALL of the following are true:

1 Your target system asserts DTACK for some area of memory.

2 AND

3 Your target system inserts at least one wait state for that area.

4 AND

5 You are going to map that area as emulation RAM or emulation ROM.

What does this do? Interlocking DTACKs simply will ensure that the emulation memory accesses to this area will have the same number of wait states as your final target does.

What will happen if ALL of the above are true and you do not interlock DTACKs? In the best case scenario, your code will run faster in the emulator than it will in your actual target. This is because an area mapped as emulation memory will always terminate with 0 wait states, even if it overlays an address where your target system inserts wait states. Interlocking DTACKs will ensure that the emulation memory accesses have the same number of wait states as your target memory.

In the worst case scenario, where you should, but do not interlock DTACKs, reads or writes to your target memory will fail. This can happen if your target system DTACK circuitry gets confused when the emulator fails to "wait" for your target system to assert DTACK.

Step 9. If emulator status shows HALTED

Most users will encounter an emulator status of HALTED at one time or another. This almost always is caused by a double-bus fault, although under rare conditions it can be caused by the target asserting the "HALT" pin. Note that the 68302 emulator NEVER asserts the HALT pin itself.

A double-bus fault occurs if the 68302 encounters an exception while processing another exception. For example, if a bus error occurs and the 68302 begins to process that bus error, then fetches an odd-address from the vector table location 08H, a double-bus fault will occur.

Most users will encounter a halted condition at least once. You can avoid this problem by making sure you have a "good" stack (that includes making sure that any needed chip-selects are programmed BEFORE any stack accesses occur) and making sure that your bus error and address error exception vector table entries (at 08H and 0CH) point to valid addresses.

If you are encountering a HALTED emulator status, you should initiate an analyzer trace before you run or step. Choose the Trace→Until Halt (ALT, T, U) command. This trace command means "trace all cycles, but NEVER trigger" and is useful because when the processor halts, the analyzer will have the last X states (based on which type analyzer card and mode you are using) in its buffer. By halting the trace with the Trace→Halt (ALT, T, H) command, you will be able to unload the trace buffer and see exactly what caused the double-bus fault.

The 68302 will halt itself if a double-bus fault occurs. Only a target system reset, or emulator reset command will clear the HALTED emulator status.

Here is an example where a Trace→Until Halt (ALT, T, U) command was used to find the cause of a double-bus fault:

```
XDEF     CS_INIT,GIMR_INIT
ORG      $0
```

```
            DC.L     $440000           ; Stack begins at $43FFFC
            DC.L     $400              ; Reset initialization code

            ORG      $400
            MOVE.W   #$0800,$F2        ; Set up the BAR for $800XXX
            MOVEA.L  #$800000,A0
            MOVE.W   #$3F02,($832,A0)  ; OR0 - 512K, read-only, 1 wait-state
            MOVE.W   #$0001,($830,A0)  ; BR0 - base address 0
            MOVE.W   #$0001,-(A7)      ; push parameter
            JSR      CHKSUM            ; jump to checksum subroutine
            MOVE.W   #$1F80,($836,A0)  ; OR1 - 256K, read-write, 0 wait-state
            MOVE.W   #$0801,($834,A0)  ; BR1 - base address $400000
            NOP
CS_INIT     NOP
            ...
```

The memory map used in this configuration includes:

```
000000..01FFFF@x eram
400000..43FFFF@x tram
```

After choosing the Trace→Until Halt (ALT, T, U) command followed by the Execution→Run... (ALT, E, R) command, selecting the User Reset option, and choosing OK, the Status window shows the emulator is HALTED. At this point, the Trace→Halt (ALT, T, H) command is chosen to stop the trace and display the results.

Looking at the trace we see:

```
state typ module\#line  :function     source
-------------------------------------------------------------------------
  -14     00040C         MOVE.W  #03F02,00832(A0)          0.640 nS
  -13     00040E           3F02  sprog rd word             0.600 nS
  -12     000410           0832  sprog rd word             0.640 nS
  -11     000412         MOVE.W  #00001,00830(A0)          0.600 nS
  -10     800832           3F02  sdata wr word         IAC 0.280 nS
   -9     000414           0001  sprog rd word             0.280 nS
   -8     000416           0830  sprog rd word             0.320 nS
   -7     000418         MOVE.W  #00001,-(A7)              0.320 nS
   -6     800830           0001  sdata wr word         IAC 0.240 nS
   -5     00041A           0001  sprog rd word             0.320 nS
   -4     00041C         JSR     0000446                   0.320 nS
   -3     00041E           0446  sprog rd word             0.320 nS
   -2     43FFFE           0001  sdata wr word             1.020 mS
   -1     43FFFC           041E  sdata wr word             1.020 mS
```

What to look for:

state -7 — A push of 0001 onto the stack is initiated. The stack pointer is already set to 44000H. The memory below there is accessed using chip-select 1, but the registers for chip-select 1 have not been initialized yet.

state -2	The processor does a write of 0001 to location 43FFFEH as a result of the instruction at -7, but because the the chip-select that would provide DTACK has not yet been initialized the cycle is not terminated normally. The 1.02mS time count is indicative of an internally generated bus error (BERR).
state -1	Because of the bus error in the previous cycle the 68302 will immediately begin to process a bus error exception. The first thing it will try to do is write the low word of the program counter value to the stack. But again, because the chip-select register has not been initialized, this stack write will also fail with a bus error, hence the double-bus fault!

This trace shows how valuable the time stamp can be, not just in highlighting cycles that are unusually long, but also in showing the effect of reprogramming of OR1. You can see how the program fetch on line -11 took approximately 600ns, but the program fetch on line -9 took only 280ns. This is a result of the write to OR1 on line -10 which changed the internal DTACK generation from 6 wait states to 1 state.

Many users capture a trace that shows the processor halting, but fail to take the time to analyze the information in the trace. Examining a trace carefully will often help you find the exact cause of a problem.

Step 10. Choose the correct target memory access size

Whenever the emulator accesses either target memory or an on-chip location, it uses the monitor program to do so. The monitor program will use either a "MOVE.B" or "MOVE.W" instruction for this access. You can control this by selecting either "8 bits" or "16 bits" for the "Target Memory Access Size" hardware configuration option.

The default value is "8 bits", and this should be acceptable for most cases. There are cases, however, where a target system design allows only word accesses to a particular area of memory, and you may wish to select "16 bits". You may also find that you need the access size set to "8 bits" sometimes and "16 bits" at other times.

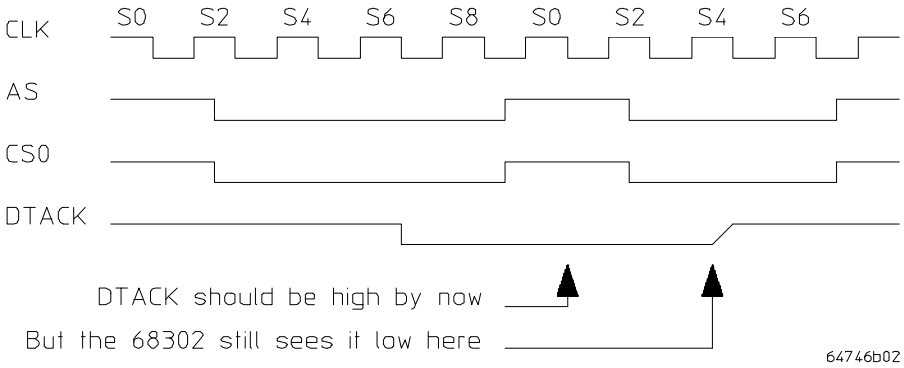
Step 11. Check your DTACK pullup resistor!

It wouldn't be fair to solve all these 68302 plug-in problems and not give the hardware engineer a chance to help out, so there's one last thing you should check before you begin.

Different from the typical 68000 family device, the DTACK signal on the 68302 is bi-directional and will be driven low by the 68302 on all cycles where DTACK is internally generated. This calls for an open-collector design in almost all cases. Have your hardware engineer check the value of the pull-up resistor used on the DTACK signal and make sure that it is approximately 1K ohms. Any value higher than 1K ohms will likely cause problems.

Lowering the value of the pullup resistor ensures that the DTACK signal rises quickly enough. If the DTACK signal rises too slowly you may see incorrect reads from target memory. Why is this? The emulator does alter the characteristics of the DTACK signal (higher capacitive loading and different drive characteristics) and can prevent the DTACK signal from rising to a logic high before the next cycle starts. If DTACK is still seen as a logic low when the next cycle starts, that cycle may be terminated with 0 wait states.

The diagram below shows how a DTACK signal that does not rise quickly enough can cause problems. On the start of the second cycle, the DTACK signal is still seen as low. Even if this is an address configured for a chip-select that has an internally generated DTACK, the slow rising external DTACK will cause problems because the 68302 will sample the DTACK as an input even on cycles programmed for internal DTACK generation. A cycle that should have been a 1 wait-state cycle terminates without any wait-states and results in bad data being read.


CLK
S0
S2
S4
S6
S8
S0
S2
S4
S6
AS
CS0
DTACK
DTACK should be high by now
But the 68302 still sees it low here
64746b02

If you have problems

Listed below are common problems and their most common causes.

Problem: You get "ERROR: Stepping failed" when trying to step.

Cause: This is most likely caused by an invalid stack. In order to step, the emulator will first modify the values on the stack, and then execute an RTE. If the stack pointer points to an invalid address, or the chip-select needed to access the stack is not initialized, the stack reads caused by the RTE will result in a double-bus fault.

Fix: Make sure the stack pointer is pointing to a legitimate address and that any chip-selects for that address are initialized.

Problem: You get "ERROR: Break failed" when trying to step.

Cause: You have configured the emulator for dedicated mode interrupts but have not put the 68302 in dedicated mode by modifying the GIMR and setting the most significant bit. Or, the instruction that you single-stepped resulted in a double-bus fault and the processor is now halted.

Fix: Make sure that the processor is set for the correct interrupt mode before single-stepping. If the interrupt mode is correct, then use the trace command before stepping so that you see what went wrong.

Problem: You get "ERROR: BERR during background access to supervisor stack" when you first try to step or run.

Cause: This problem occurs when the emulator attempts to modify the stack so that your run or step command will begin from the proper address. The emulator will try to modify the stack and then use an RTE to force the program counter and status register to the proper values. In this case, a bus error occurred when the emulator attempted to modify the stack.

Fix: Make sure the stack pointer is pointing to a legitimate address and that any chip-selects for that address are initialized.

Problem: You get "ERROR: Monitor failure; bus error" when trying to do a display, modify, or load command.

Cause: This error means that when the emulator tries to read from, or write to, a memory location mapped as target RAM or ROM, a bus error exception occurred. This can have many causes, but most often is caused when you try to access memory that relies on a chip-select signal, but have not initialized the chip-select registers first.

Fix: Make sure that your chip-select registers are properly initialized.

Problem: Whenever you break into the monitor you have problems with the SCC portion of your program.

Cause: The emulator asserts the FRZ pin when it breaks into the monitor and this in turn "freezes" the Communications Processor. Based on what type of SCC setup you have, and what type of activity is occurring at the time of the break, you may experience unexpected SCC errors or activity. Note that if you are using the background monitor the FRZ is asserted during the entire time the emulator is running in the monitor. If you are using the foreground monitor, the FRZ pin is asserted for a short time during transition from your program to the monitor program (approximately 30 CPU cycles).

Fix: Either use the foreground monitor, which will reduce but not eliminate the time that the FRZ pin is asserted, or avoid setting any breakpoints while debugging SCC functions. By using the trace analyzer, you can capture a real-time log of SCC-related bus activity.

Problem: You are using the DRAM refresh controller of the 68302, but when you break into the monitor, the values in your DRAM become corrupted.

Cause: You are either using the background monitor, or have not made the proper modification to the foreground monitor.

Fix: Use the foreground monitor and include the necessary modifications. Refer to "Step 7. If you use the 68302 built-in DRAM refresh" for details on how to ensure proper DRAM refresh.

Problem: The emulator status shows HALTED

Cause: A double-bus fault has occurred, or the target system has asserted the HLT pin. This is almost always caused by a double-bus fault. Most often, a bus error or address error exception fails to complete correctly because of an invalid stack or vector table entry.

Fix: Make sure you have a valid stack and valid vector table entries. Refer to "Step 9. If emulator status shows HALTED" for troubleshooting information.

Problem: Incorrect data is read from target RAM or ROM.

Cause: This is most often caused by a slow rising DTACK signal. The DTACK signal is usually pulled high with a resistor. Any resistor value above 1K ohms will usually result in some target memory cycles that are terminated with 0 wait states regardless of how the chip-select registers are programmed.

Fix: Make sure that the pullup resistor on DTACK is no higher than 1K ohms. Refer to "Step 11. Check your DTACK pullup resistor!" for more information.

5

Debugging Programs

Debugging Programs

This chapter contains information on loading and debugging programs.

- Loading and Displaying Programs
- Displaying Symbol Information
- Stepping, Running, and Stopping the Program
- Using Breakpoints and Break Macros
- Displaying and Editing Variables
- Displaying and Editing Memory
- Displaying and Editing I/O Locations
- Displaying and Editing Registers
- Making Coverage Measurements
- Tracing Program Execution
- Setting Up Custom Trace Specifications

Loading and Displaying Programs

This section shows you how:

- To load user programs
- To display source code only
- To display source code mixed with assembly instructions
- To display source files by their names
- To specify source file directories
- To search for function names in the source files
- To search for addresses in the source files
- To search for strings in the source files

To load user programs

1. Choose the File→Load Object... (ALT, F, L) command.
2. Select the function code of the memory space into which the program should be loaded.
3. Select the file to be loaded.
4. Choose the Load button to load the program.

Programs are only loaded into the memory ranges mapped with the same *function code*.

With this command, you can load any IEEE-695 object file created with any of the Microtec or HP programming tools for 68000.

To display source code only

1. Position the cursor on the starting line to be displayed.
2. From the Source window control menu, choose the Display→Source Only (ALT, -, D, S) command.

   The Source window may be toggled between the C source only display and the C source/mnemonic mixed display.

   The display starts from the line containing the cursor.

   The source only display shows line numbers with the source code.

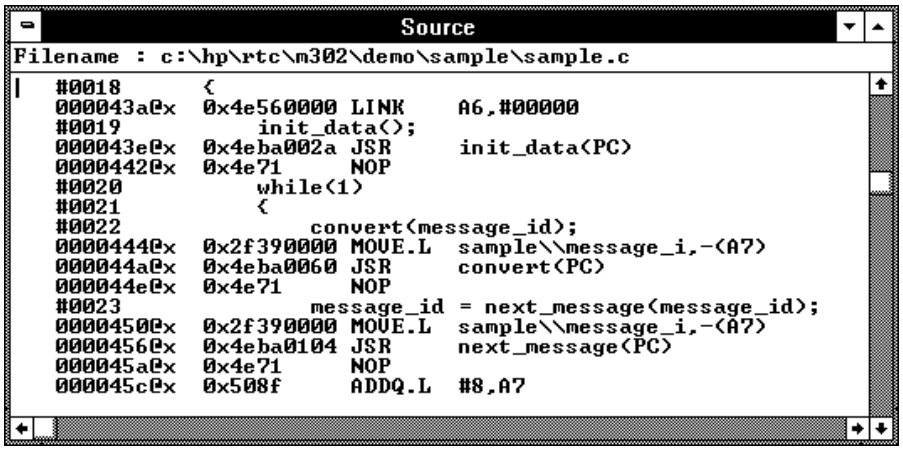
To display source code mixed with assembly instructions

1. Position the cursor on the starting line to be displayed.
2. From the Source window control menu, choose the Display→Mixed Mode (ALT, -, D, M) command.

   The mnemonic display contains the address, data, and disassembled instruction mnemonics intermixed with the C source lines.

Example

C Source/Mnemonic Mode Display



To display source files by their names

1. Make the Source window the active window, and choose the Display→Select Source... (ALT, -, D, L) command from the Source window's control menu.
2. Select the desired file.
3. Choose the Select button.
4. Choose the Close button.

Note

The contents of assembly language source files cannot be displayed.

To specify source file directories

1. Make the Source window the active window, and choose the Display→Select Source... (ALT, -, D, L) command from the Source window's control menu.
2. Choose the Directory... button.
3. Enter the directory name in the Directory text box.
4. Choose the Add button.
5. Choose the Close button to close the Search Directories dialog box.
6. Choose the Close button to close the Select Source dialog box.

If the source files associated with the loaded object file are in different directories from the object file, you must identify the directories in which the source files can be found.

You can also specify them source file directories by setting the SRCPATH environment variable in MS-DOS as follows:

```
set SRCPATH=<full path 1>;<full path 2>
```

To search for function names in the source files

1. From the Source window's control menu, choose the Search→Function... (ALT, -, R, F) command.
2. Select the function to be searched.
3. Choose the Find button.
4. Choose the Close button.

Disassembled instructions are displayed in the Source window for assembly language source files.

To search for addresses in the source files

1. From the Source window's control menu, choose the Search→Address... (ALT, -, R, A) command.
2. Type or paste the address into the Address text box.
3. Choose the Find button.
4. Choose the Close button.

Disassembled instructions are displayed in the Source window for assembly language source files.

To search for strings in the source files

1. From the Source window's control menu, choose the Search→String... (ALT, -, R, S) command.
2. Type or paste the string into the String text box.
3. Select whether the search should be case sensitive.
4. Select whether the search should be down (forward) or up (backward).
5. Choose the Find Next button. Repeat this step to search for the next occurrence of the string.
6. Choose the Cancel button to close the dialog box.

Displaying Symbol Information

This section shows you how:

- To display program module information
- To display function information
- To display external symbol information
- To display local symbol information
- To display global assembler symbol information
- To display local assembler symbol information
- To create a user-defined symbol
- To display user-defined symbol information
- To delete a user-defined symbol
- To display the symbols containing the specified string

To display program module information

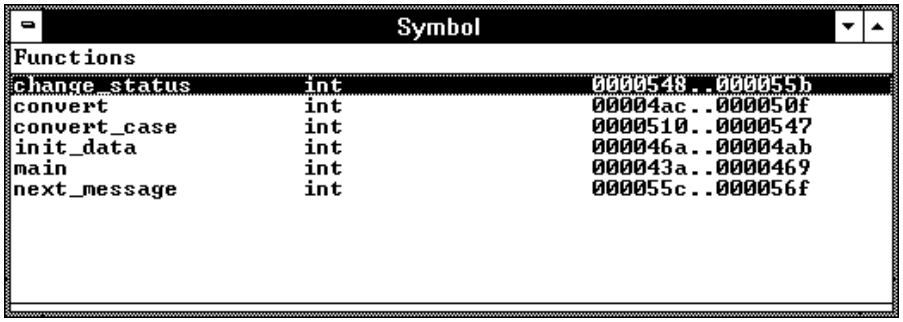
- From the Symbol window's control menu, choose the Display→Modules (ALT, -, D, M) command.

To display function information

- From the Symbol window's control menu, choose the Display→Functions (ALT, -, D, F) command.

  The name, type, and address range for the functions in the program are displayed.

Example Function Information Display



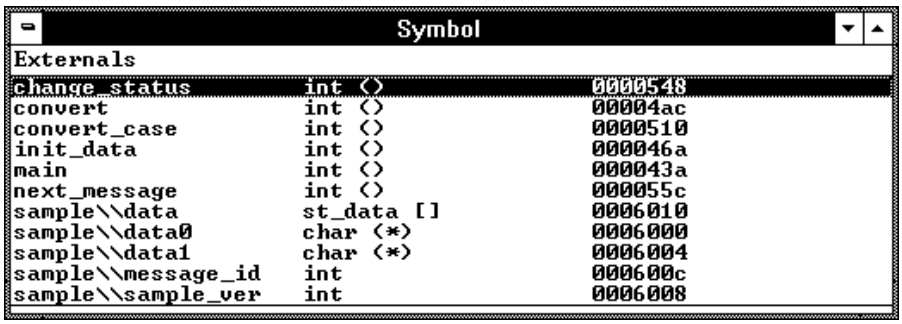
To display external symbol information

- From the Symbol window's control menu, choose the Display→Externals (ALT, -, D, E) command.

  The name, type, and address of the global variables in the program are displayed.

Example

External Symbol Information Display

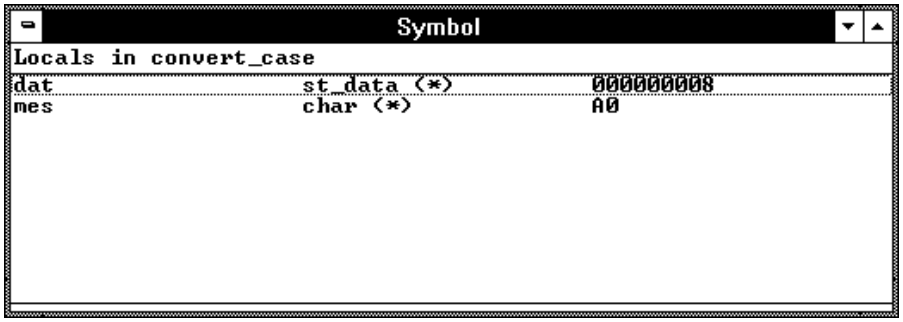


To display local symbol information

1. From the Symbol window's control menu, choose the Display→Locals... (ALT, -, D, L) command.
2. Type or paste the function for which the local variable information is to displayed.
3. Choose the OK button.

The name, type, and offset from the stack frame of the local variables in the selected function are displayed.

Example Local Symbol Information Display



To display global assembler symbol information

- From the Symbol window's control menu, choose the Display→Asm Globals (ALT, -, D, G) command.

  The name and address for the global assembler symbols in the program are displayed.

To display local assembler symbol information

1. From the Symbol window's control menu, choose the Display→Asm Locals... (ALT, -, D, A) command.
2. Type or paste the module for which the local variable information is displayed.
3. Choose the OK button.

   The name and address for the local assembler variables in the selected module are displayed.

To create a user-defined symbol

1 From the Symbol window's control menu, choose the User defined→Add... (ALT, -, U, A) command.

2 Type the symbol name in the Symbol Name text box.

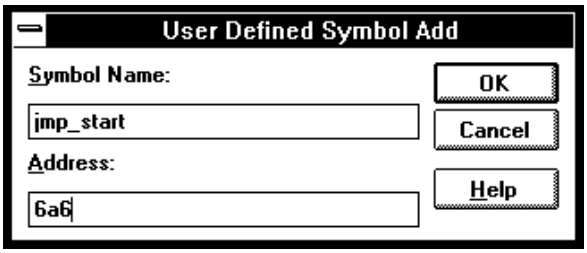
3 Type the address in the Address text box.

4 Choose the OK button.

User-defined symbols, just as standard symbols, can be used as address values when entering commands.

Example

To add the user-defined symbol "jmp_start":



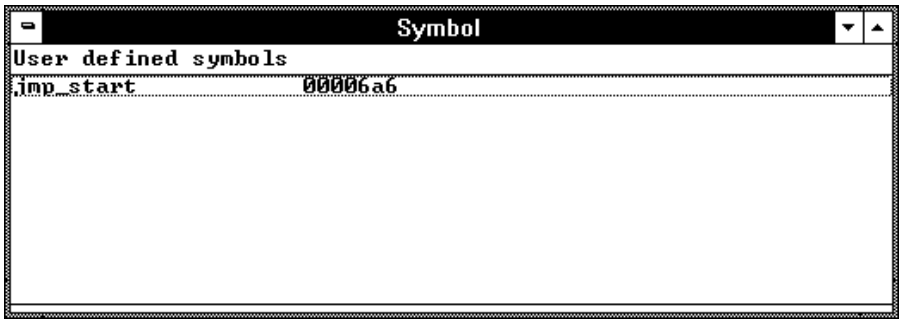
To display user-defined symbol information

- From the Symbol window's control menu, choose the Display→User defined (ALT, -, D, U) command.

The command displays the name and address for the user-defined symbols.

Example

User-Defined Symbol Information Display



To delete a user-defined symbol

1. From the Symbol window's control menu, choose the Display→User defined (ALT, -, D, U) command to display the user-defined symbols.
2. Select the user-defined symbol to be deleted.
3. From the Symbol window's control menu, choose the User defined→Delete (ALT, -, U, D) command.

To display the symbols containing the specified string

1. From the Symbol window's control menu, choose the FindString→String... (ALT, -, F, S) command.
2. Type or paste the string in the String text box. The search will be case-sensitive.
3. Choose the OK button.

   To restore the original non-selective display, redisplay the symbolic information.

Stepping, Running, and Stopping the Program

This section shows you how:

- To step a single line or instruction
- To step over a function
- To step multiple lines or instructions
- To run the program until the specified line
- To run the program until the current function return
- To run the program from a specified address
- To stop program execution
- To reset the processor

To step a single line or instruction

- Choose the Execution→Single Step (ALT, E, N) command.
- Or, press the F2 key.

In the source display mode, this command executes the C source code line at the current program counter address.

In the source/mnemonic mixed display mode, the command executes the microprocessor instruction at the current program counter address.

Once the source line or instruction has executed, the next program counter address highlighted.

To step over a function

- Choose the Execution→Step Over (ALT, E, O) command.
- Or, press the F3 key.

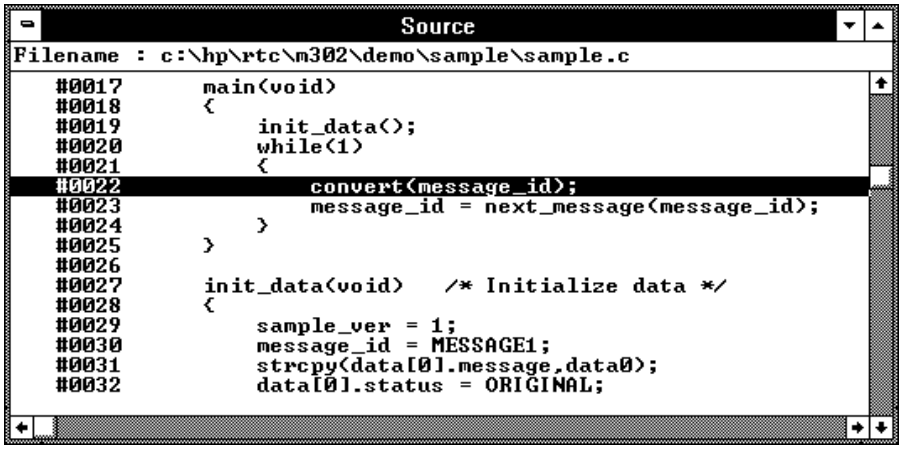
This command steps a single source line or assembly language instruction except when the source line contains a function call or the assembly instruction makes a subroutine call. In these cases, the entire function or subroutine is executed.

In the source/mnemonic mixed display mode, the command does not distinguish between the following two types of instructions:

JSR

BSR

Example



When the current program counter is at line 22, choosing the Execution→Step Over (ALT, E, O) command steps over the "convert" function. Once the function has been stepped over, the program counter indicates line 23.

To step multiple lines or instructions

1. Choose the Execution→Step... (ALT, E, S) command.
2. Select one of the Current PC, Start Address, or Address options. (Enter the starting address when the Address option is selected.)
3. In the Count text box, type the number of lines to be single-stepped.
4. Choose the Execute button.
5. Choose the Close button to close the dialog box.

The Current PC option starts single-stepping from the current PC address. The Start Address option starts single-stepping from the *transfer address*. The Address option starts single-stepping from the address specified in the text box.

In the source only display mode, the command steps the number of C source lines specified. In the source/mnemonic mixed display mode, the command steps the number of microprocessor instructions specified.

When the step count specified in the Count text box is 2 or greater, the count decrements by one as each line or instruction executes. A count of 1 remains in the Count text box. Also, in the Source window, the highlighted line that indicates the current program counter moves for each step.

To step over functions, select the Over check box.

To run the program until the specified line

1 Position the cursor in the Source window on the line that you want to run to.

2 Choose the Execution→Run to Cursor (ALT, E, C) command.

Execution stops immediately before the cursor-selected line.

Because this command uses breakpoints, you cannot use it when programs are stored in target system ROM.

If the specified address is not reached within the number of milliseconds specified by StepTimerLen in the B3621.INI file, a dialog box appears, asking you to cancel the command by choosing the Stop button. When the Stop button is chosen, the program execution stops, the breakpoint is deleted, and the processor transfers to the RUNNING IN USER PROGRAM status.

Note

This can be done more quickly by using the pop-up menu available with the right mouse button.

To run the program until the current function return

- Choose the Execution→Run to Caller (ALT, E, T) command.

The Execution→Run to Caller (ALT, E, T) command executes the program from the current program counter address up to the return from the current function.

Note

The debugger cannot properly run to the function return when the current program counter is at the first line of the function (immediately after its entry point). Before running to the caller, use the Execution→Single Step (ALT, E, N) command to step past the first line of the function.

To run the program from a specified address

1. Choose the Execution→Run... (ALT, E, R) command.
2. Select one of the Current PC, Start Address, User Reset, or Address options. (Enter the address when the Address option is selected.)
3. Choose the Run button.

The Current PC option executes the program from the current program counter address. The Start Address option executes the program from the *transfer address*.

The User Reset option initiates program execution on receiving a RESET signal from the target system. The reset wait status can be cleared with the Execution→Reset (ALT, E, E) command.

The Address option executes the program from the address specified.

To stop program execution

- Choose the Execution→Break (ALT, E, B) command, or press the F4 key.

As soon as the Execution→Break (ALT, E, B) command is chosen, the emulator starts running in the monitor.

To reset the processor

- Choose the Execution→Reset (ALT, E, E) command.

  Once the command has been completed, the processor remains reset if monitor intrusion is disallowed. If monitor intrusion is allowed, the emulation microprocessor may switch immediately from reset to running in monitor, for example, to update the contents of a register window.

  If a foreground monitor is selected, it will automatically be loaded when this command is executed. This is done to make sure the foreground monitor code is intact.

Using Breakpoints and Break Macros

This section shows you how:

- To set a breakpoint
- To disable a breakpoint
- To delete a single breakpoint
- To list the breakpoints and break macros
- To set a break macro
- To delete a single break macro
- To delete all breakpoints and break macros

A breakpoint is an address you identify in the user program where program execution is to stop. Breakpoints let you look at the state of the target system at particular points in the program.

A break macro is a breakpoint followed by any number of macro commands (which are the same as command file commands).

Because breakpoints are set by replacing opcodes in the program, you cannot set breakpoints or break macros in programs stored in target system ROM.

All breakpoints are deleted when RTC is exited.

To set a breakpoint

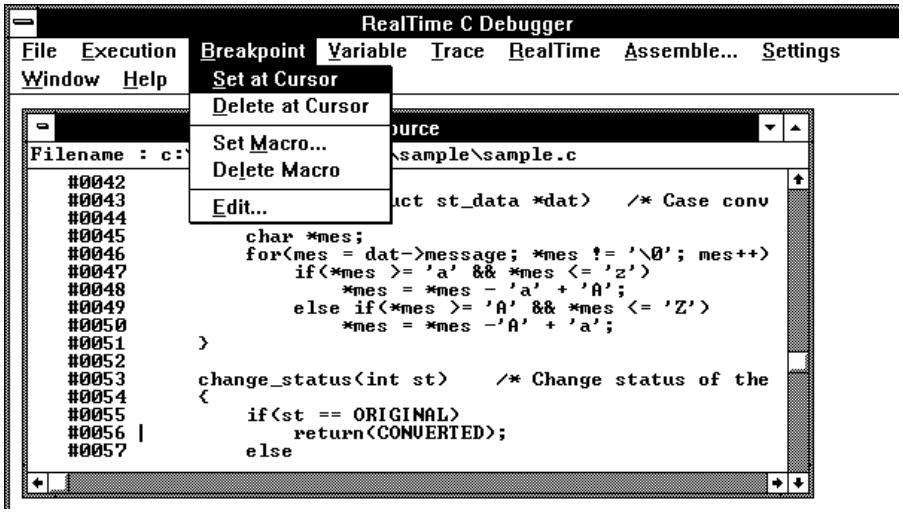
1 Position the cursor on the line where you wish to set a breakpoint.

2 Choose the Breakpoint→Set at Cursor (ALT, B, S) command.

When you run the program and the breakpoint is hit, execution stops immediately before the breakpoint line. The current program counter location is highlighted.

Example

To set a breakpoint at line 56:



Note

This can be done more quickly by using the pop-up menu available with the right mouse button.

To disable a breakpoint

1. Choose the Breakpoint→Edit... (ALT, B, E) command.
2. Select the breakpoint to be disabled.
3. Choose the Enable/Disable button. Notice that "DI" appears next to the breakpoint in the list.
4. To close the dialog box, choose the Close button.

You can reenable a breakpoint in the same manner by choosing the Breakpoint→Edit... (ALT, B, E) command, selecting a disabled breakpoint from the list, and choosing the Enable/Disable Button.

To delete a single breakpoint

- Position the cursor on the line that has the breakpoint to be deleted, and choose the Breakpoint→Delete at Cursor (ALT, B, D) command.

Or:

1. Choose the Breakpoint→Edit... (ALT, B, E) command.
2. Select the breakpoint to be deleted.
3. Choose the Delete button.
4. Choose the Close button.

The Breakpoint→Edit... (ALT, B, E) command allows you to delete all the breakpoints and break macros at once with the Delete All button.

To list the breakpoints and break macros

- Choose the Breakpoint→Edit... (ALT, B, E) command.

The command displays breakpoints followed by break macro commands in parentheses.

The Breakpoint Edit dialog box also allows you to delete breakpoints and break macros.

To set a break macro

1. Position the cursor on the line where you wish to set a break macro.
2. Choose the Breakpoint→Set Macro... (ALT, B, M) command.
3. Select the Add Macro check box in the Breakpoint Edit dialog box.
4. Specify the macro command in the Macro Command text box.
5. Choose the Set button.
6. To add another macro command, repeat steps 4 and 5.
7. To exit the Breakpoint Edit dialog box, choose the Close button.

The debugger automatically executes the specified macro commands when the *break macro* line is reached.

To add macro commands after an existing macro command, position the cursor on the macro command before choosing Breakpoint→Set Macro... (ALT, B, M).

To add macro commands to the top of an existing break macro, position the cursor on the line that contains the BP marker before choosing Breakpoint→Set Macro... (ALT, B, M).

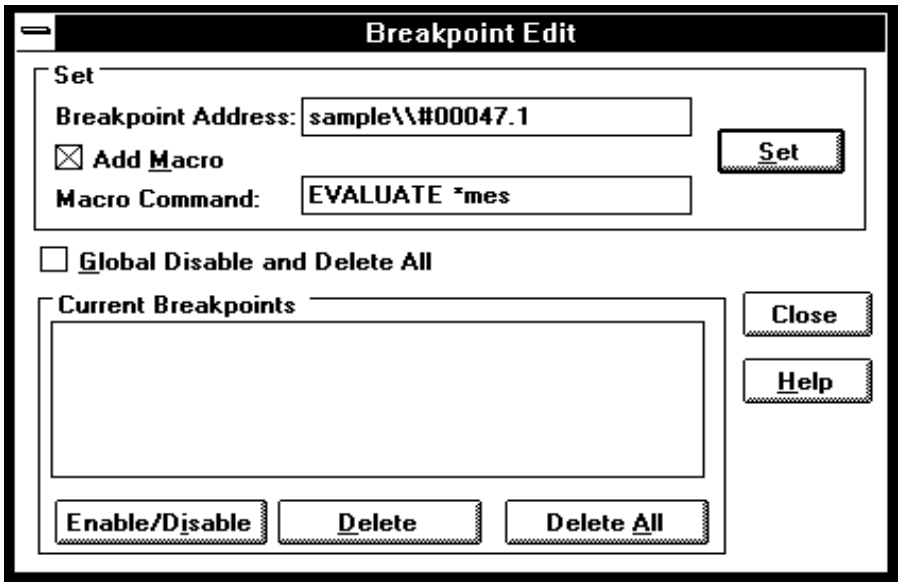
Example

To set "EVALUATE" and "RUN" break macros:

Position the cursor on line 47; then, choose the Breakpoint→Set Macro... (ALT, B, M) command.

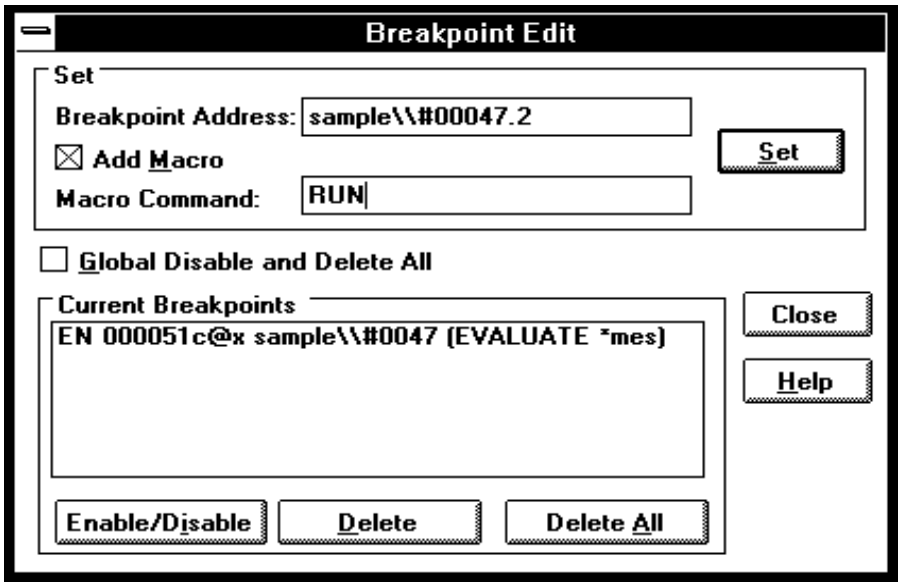
Select the Add Macro check box.

Enter "EVALUATE *mes" in the Macro Command text box.



Choose the Set button.

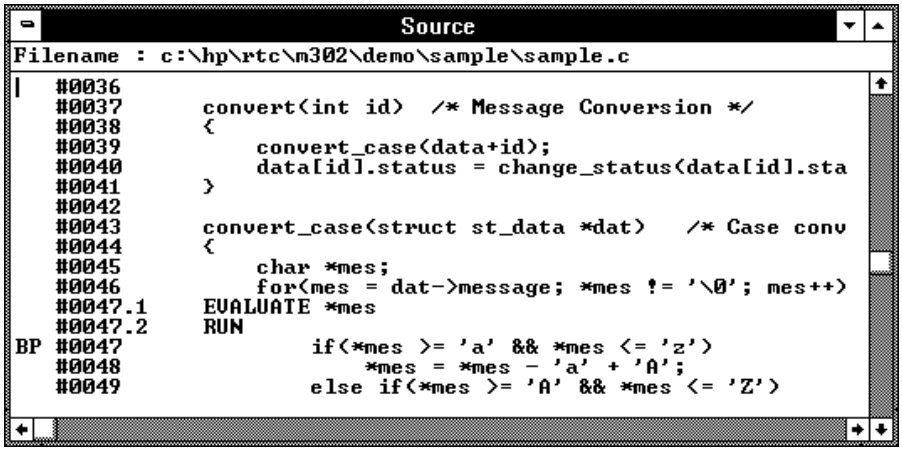
Enter "RUN" in the Macro Command text box.



Choose the Set button.

Choose the Close button.

The break macro is displayed in the Source window as shown below.



To delete a single break macro

1. Position the cursor on the line that contains the break macro to be deleted.

2. Choose the Breakpoint→Delete Macro (ALT, B, L) command.

   To delete a single macro command that is part of a break macro, position the cursor on the macro command before choosing Breakpoint→Delete Macro (ALT, B, L).

   The Breakpoint→Edit... (ALT, B, E) command allows you to delete all the breakpoints and break macros at once by choosing the Delete All button. Also, by selecting the Global Disable and Delete All check box, you can delete all breakpoints and break macros and prevent creation of new breakpoints and break macros.

To delete all breakpoints and break macros

1. Choose the Breakpoint→Edit... (ALT, B, E) command.
2. Choose the Delete All button.
3. Select the Global Disable and Delete All check box.
4. Choose the Close button.

The Breakpoint→Edit... (ALT, B, E) command allows you to delete all the breakpoints and break macros at once with the Delete All button. Also, you can delete all breakpoints and break macros and prevent creation of new breakpoints and break macros by selecting the Global Disable and Delete All check box.

Displaying and Editing Variables

This section shows you how:

- To display a variable
- To edit a variable
- To monitor a variable in the WatchPoint window

To display a variable

1. Position the mouse pointer over the variable in the Source window and double-click the left mouse button.
2. Choose the Variable→Edit... (ALT, V, E) command.
3. Choose the Update button to read the contents of the variable and display the value in the dialog box.
4. To exit the Variable dialog box, choose the Close button.

Note that you can update the contents of an auto variable only while the program executes within the scope function.

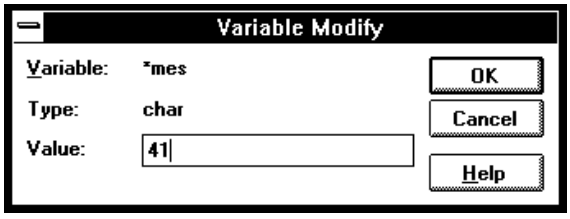
To edit a variable

1 Position the mouse pointer over the variable in the Source window and double-click the left mouse button.

2 Choose the Variable→Edit... (ALT, V, E) command.

3 Choose the Modify button. This opens the Variable Modify dialog box.

4 Type the desired value in the Value text box. The value must be of the type specified in the Type field.



5 Choose the OK button.

6 Choose the Close button.

Note that you can change the contents of an auto variable only while the program executes within the scope function.

To monitor a variable in the WatchPoint window

1. Highlight the variable in the Source window by either double-clicking the left mouse button or by holding the left mouse button down and dragging the mouse pointer over the variable.
2. Choose the Variable→Edit... (ALT, V, E) command.
3. Choose the "to WP" button.
4. Choose the Close button.
5. To open the WatchPoint window, choose the Window→WatchPoint command.

   Note that you can only monitor an auto variable in the WatchPoint window when the program executes within the scope function.

Displaying and Editing Memory

This section shows you how:

- To display memory
- To edit memory
- To copy memory to a different location
- To copy target system memory into emulation memory
- To modify a range of memory with a value
- To search memory for a value or string

To display memory

1. Choose the RealTime→Memory Polling→ON (ALT, R, M, O) command.
2. Choose the Window→Memory command.
3. Double-click one of the addresses.
4. Use the keyboard to enter the address of the memory locations to be displayed.
5. Press the Enter key.

   An address may be entered as a value or symbol. You can also select the desired address by using the scroll bar.

   To change the size of the data displayed, access the Memory window's control menu; then, choose the Display→Byte (ALT, -, D, Y), Display→16 Bits (ALT, -, D, 1), or Display→32 Bits (ALT, -, D, 3) command. When the

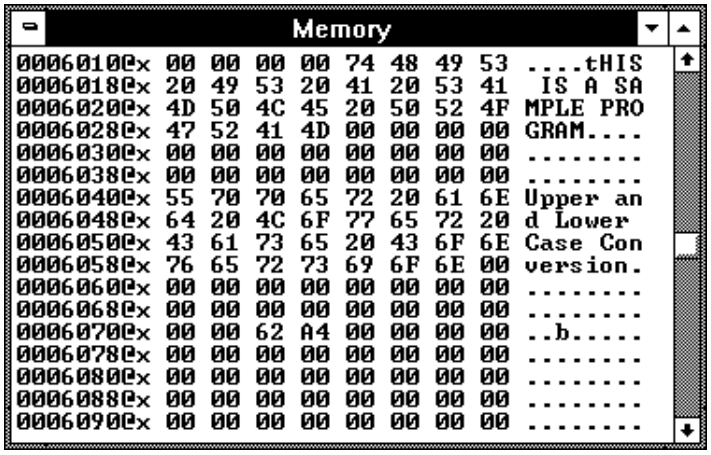
Display→Byte (ALT, -, D, Y) command is chosen, ASCII values are also displayed.

To specify whether memory is displayed in a single-column or multicolumn format, access the Memory window's control menu; then, choose the Display→Linear (ALT, -, D, L) or Display→Block (ALT, -, D, B) command. When the Display→Linear (ALT, -, D, L) command is chosen, symbolic information associated with an address is also displayed.

The Memory window display is updated periodically. When the window displays the contents of target system memory, user program execution is temporarily suspended as the display is updated. To prevent program execution from being temporarily suspended (and the Memory window from being updated), choose the RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command to activate the real-time mode.

Example

Memory Displayed in Byte Format



To edit memory

Assuming the location you wish to edit has already been displayed (and Memory window polling is turned ON):

1. Double-click the location you wish to edit.
2. Use the keyboard to enter a new value.
3. Press the Enter key. Notice that the next location is highlighted.
4. Repeat steps 2 and 3 to edit successive locations.

Editing the contents of target system memory causes user program execution to be temporarily interrupted. You cannot modify the contents of target memory when the emulator is running the user program and monitor intrusion is disallowed.

To copy memory to a different location

1. From the Memory window's control menu, choose the Utilities→Copy... (ALT, -, U, C) command.
2. Enter the starting address of the range to be copied in the Start text box.
3. Enter the end address of the range to be copied in the End text box.
4. Enter the address of the destination in the Destination text box.
5. Choose the Execute button.
6. To close the Memory Copy dialog box, choose the Close button.

To copy target system memory into emulation memory

1 Map the address range to be copied as emulation memory.

2 Because the processor cannot read target system memory when it is in the EMULATION RESET state, choose the Execution→Break (ALT, E, B) command, or press the F4 key, to break execution into the monitor.

3 From the Memory window's control menu, choose the Utilities→Image... (ALT, -, U, I) command.

4 Enter the starting address in the Start text box.

5 Enter the end address in the End text box.

6 Choose the Execute button.

7 To exit the Memory Image Copy dialog box, choose the Close button.

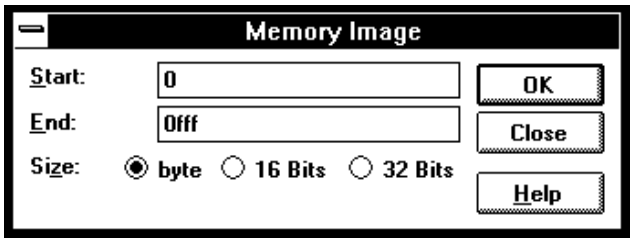
This command is used to gain access to features that are available with emulation memory (like breakpoints).

The following commands cannot be used when programs are stored in target system ROM. However, you can use these commands if you copy the contents of target system ROM into emulation memory with the Utilities→Image... (ALT, -, U, I) command:

Breakpoint→Set at Cursor (ALT, B, S)
Breakpoint→Delete at Cursor (ALT, B, D)
Breakpoint→Set Macro... (ALT, B, M)
Breakpoint→Delete Macro (ALT, B, L)
Execution→Run to Cursor (ALT, E, C)
Execution→Run to Caller (ALT, E, T)
Settings→Coverage→Coverage ON (ALT, S, V, O)
Settings→Coverage→Coverage Reset (ALT, S, V, R)

Example

To copy the contents of addresses 0h through ffffh with "X" function code from target system memory to the corresponding emulation memory address range:



To modify a range of memory with a value

1. From the Memory window's control menu, choose the Utilities→Fill... (ALT, -, U, F) command.
2. Enter the desired value in the Value text box.
3. Enter the starting address of the memory range in the Start text box.
4. Enter the end address in the End text box.
5. Select one of the Size options.
6. Choose the Execute button.

   The Byte, 16 Bit, or 32 Bit size option specifies the size of the values that are used to fill memory.

To search memory for a value or string

1 From the Memory window's control menu, choose the Search... (ALT, -, R) command.

2 Enter in the Value or String text box the value or string to search for.

3 Enter the starting address in the Start text box.

4 Enter the end address in the End text box.

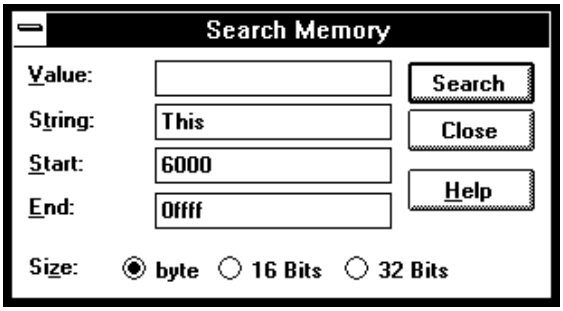
5 Choose the Execute button.

6 Choose the Close button.

When the specified data is found, the location at which the value or string was found is displayed in the Memory window.

Example

To search addresses 6000h through 0fffffh, for the string "This":



Displaying and Editing I/O Locations

This section shows you how:

- To display I/O locations
- To edit an I/O location

With the 68302 microprocessor, I/O locations are memory-mapped.

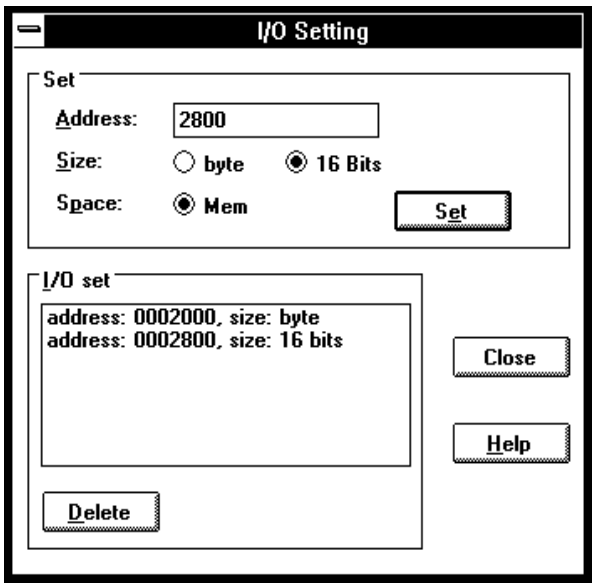
To display I/O locations

1. Choose the Window→I/O command.
2. From the I/O window's control menu, choose the Define... (ALT, -, D) command.
3. Enter the address in the Address text box.
4. Select whether the size of the I/O location is a Byte or 16 Bits.
5. Choose the Set button.
6. Choose the Close button.

The Window→I/O command displays the contents of the specified I/O locations.

The debugger periodically reads the I/O locations and displays the latest status in the I/O window. To prevent the debugger from reading the I/O locations (and updating the I/O window), choose the RealTime→I/O Polling→OFF (ALT, R, I, F) command.

Example To display the contents of address 2000:



To edit an I/O location

1. Display the I/O value to be changed with the Window→I/O command.
2. Double-click the value to be changed.
3. Use the keyboard to enter a new value.
4. Press the Enter key.

To confirm the modified values, press the Enter key for every changed value.

Editing the I/O locations temporarily halts user program execution. You cannot modify I/O locations while the user program executes in the real-time mode or when I/O polling is turned OFF.

Displaying and Editing Registers

This section shows you how:

- To display registers
- To edit registers

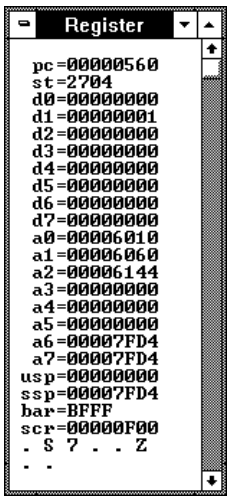
To display registers

- Choose the Window→Register command.

The register values displayed in the window are periodically updated to show you how the values change during program execution. The decoded flag register flags allow you to identify the register status at a glance.

When the Register window is updated, user program execution is temporarily interrupted. To prevent the user program from being interrupted (and the Register window from being updated), choose the RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command to activate the real-time mode.

Example Register Contents Displayed in the Register Window



To edit registers

1. Display the register contents by choosing the Window→Register command.
2. Double-click the value to be changed.
3. Use the keyboard to enter a new value.
4. Press the Enter key.

Modifying register contents temporarily interrupts program execution. You cannot modify register contents while the user program is running and monitor intrusion is disallowed.

Note that register values are not actually changed until the Enter key is pressed.

Double-clicking the status register (st) contents opens the Register Bit Fields dialog box which you can use to set or clear individual bit fields.

Making Coverage Measurements

This section shows you how:

- To display execution coverage

Coverage measurements are not supported when using the HP 64170 emulation memory board.

To display execution coverage

Coverage measurements are not supported when using the HP 64170 emulation memory board.

1. Choose the Settings→Coverage→Coverage Reset (ALT, S, V, R) command.
2. Execute the user program.
3. Choose the Settings→Coverage→Coverage ON (ALT, S, V, O) command.

   This command checks and displays the program execution coverage. The coverage display highlights the statements fetched since the last coverage reset.

   If you display execution coverage without resetting the previous execution coverage, the measurement will not be correct.

   In addition, execution coverage can only be displayed for programs stored in emulation memory. To display execution coverage for programs stored in target system memory, first transfer the program into emulation memory.

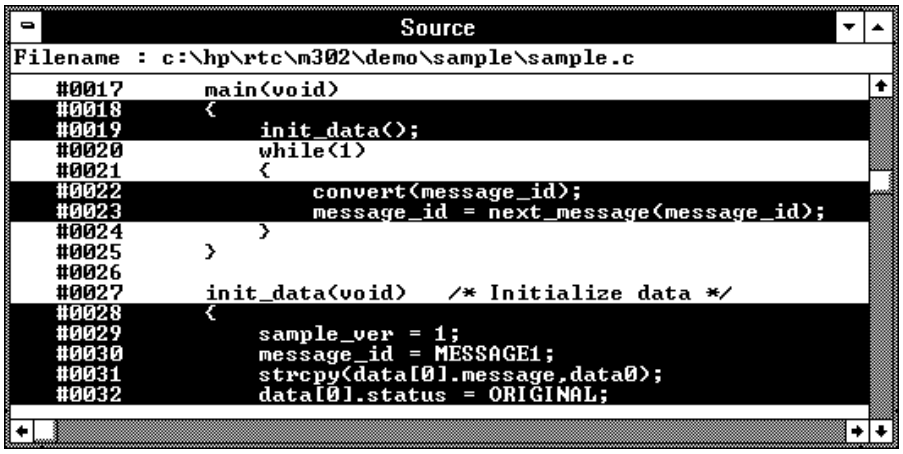
   To hide the execution coverage data, choose the Settings→Coverage→Coverage OFF (ALT, S, V, F) command.

Note

The coverage display also highlights a source symbol when the source symbol corresponds to a single assembly language instruction and the instruction is prefetched.

Example

Execution Coverage Displayed in Source Window



Tracing Program Execution

This section shows you how:

- To trace function flow
- To trace callers of a specified function
- To trace execution within a specified function
- To trace accesses to a specified variable
- To trace before a particular variable value and break
- To trace until the command is halted
- To stop a running trace
- To repeat the last trace
- To identify bus arbitration cycles in the trace
- To display bus cycles
- To display absolute or relative counts

How the Analyzer Works

When you trace program execution, the analyzer captures microprocessor address bus, data bus, and control signal values at each clock cycle. The values captured for one clock cycle are collectively called a state. A trace is a collection of these states stored in analyzer memory (also called trace memory).

The trigger condition tells the analyzer when to store states in trace memory. The trigger position specifies whether states are stored before, after, or about the state that satisfies the trigger condition.

The store condition limits the kinds of states that are stored in trace memory.

When the states stored are limited by the store condition, up to two states which satisfy the prestore condition may be stored when they occur before the states that satisfy the store condition.

After a captured state satisfies the trigger condition, a trace becomes complete when trace memory is filled with states that satisfy the store and prestore conditions.

Note

The analyzer traces unexecuted instructions due to prefetching in 68302.

Trace Window Contents

When traces are completed, the Trace window is automatically opened to display the trace results.

Each line in the trace shows the trace buffer state number, the type of state, the module name and line number, the function name, the source file information, and the time information for the state (relative to the other lines, by default).

When bus cycles are included, the address, data, and disassembled instruction or bus cycle status mnemonics are shown.

To trace function flow

- Choose the Trace→Function Flow (ALT, T, F) command.

The command stores function entry points, and the resulting trace shows program execution flow.

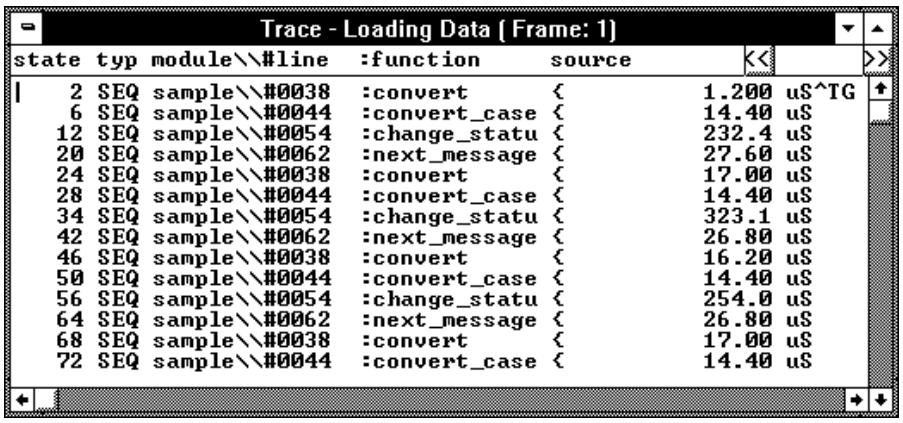
The command traces C function entry points only. It does not trace execution for assembly language routines.

Note

When using the MCC68K compiler, you must specify the -Kf option when compiling programs in order for the debugger to be able to trace function flow.

Example

Function Flow Trace



To trace callers of a specified function

1. Double-click the function name in one of the debugger windows.
2. Choose the Trace→Function Caller... (ALT, T, C) command.
3. Choose the OK button.

This command stores the first executable statement of the specified function and prestores statements that execute before it. The prestored statements show the caller of the function.

To identify interrupts in program execution, trace the caller of the interrupt process routine using the Trace→Function Caller... (ALT, T, C) command.

For Assembler symbols, the system traces the last two instructions executed before the specified Assembler symbol is reached. Specifying the first symbol of a subroutine enables the system to trace the caller of the subroutine.

Note

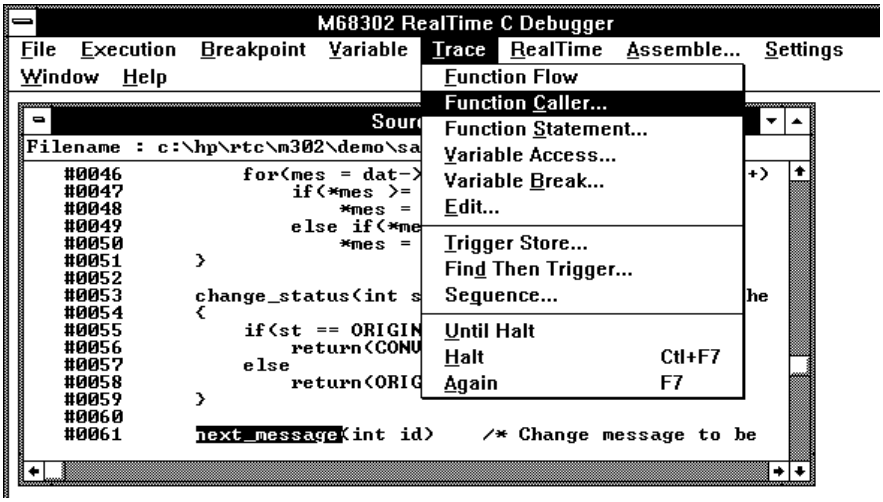
The analyzer may fail in tracing the caller due to prefetching in 68302. To avoid this failure, specify the function by a value of its address + 2.

Example

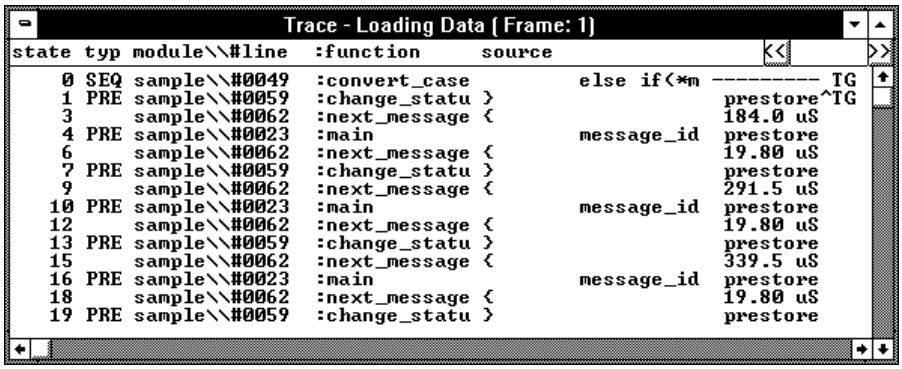
To trace the caller of "next_message":

Double-click "next_message".

Choose the Trace→Function Caller... (ALT, T, C) command.



The Trace window becomes active and displays the trace results.



You can see how prefetching affects tracing by choosing the Display→Mixed Mode (ALT, -, D, M) command from the Trace window's control menu.

To trace execution within a specified function

1 Double-click the function name in the Source window.

2 Choose the Trace→Function Statement... (ALT, T, S) command.

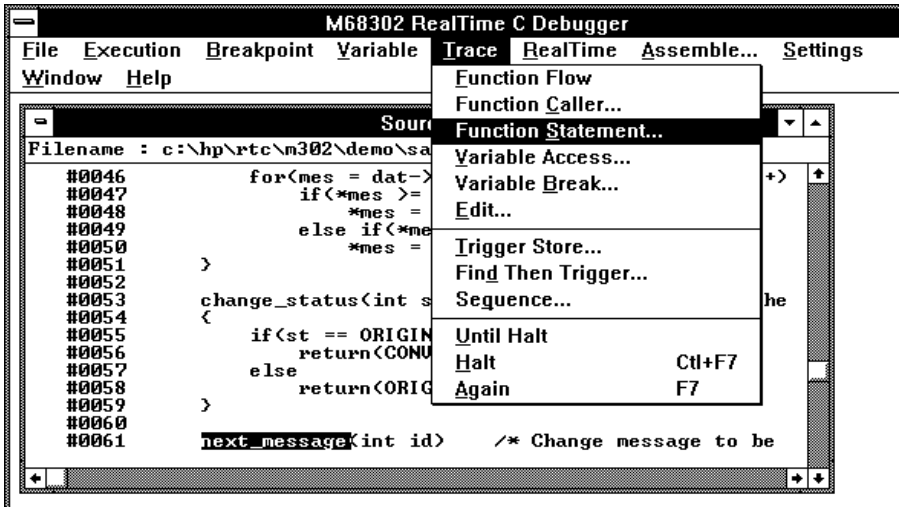
This command traces C functions only. It does not trace execution of assembly language subroutines.

Example

To trace execution within "next_message":

Double-click "next_message".

Choose the Trace→Function Statement... (ALT, T, S) command.



The Trace window becomes active and displays the results. You can see how prefetching affects tracing by choosing the Display→Mixed Mode (ALT, -, D, M) command from the Trace window's control menu.

To trace accesses to a specified variable

1 Double-click the global variable name in the Source window.

2 Choose the Trace→Variable Access... (ALT, T, V) command.

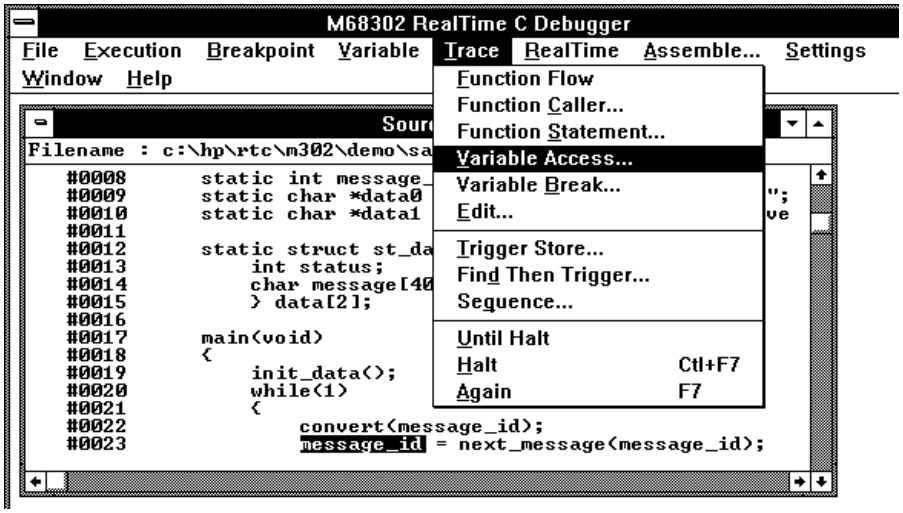
The command also traces access to the Assembler symbol specified by its name and size.

Example

To trace access to "message_id":

Double-click "message_id".

Choose the Trace→Variable Access... (ALT, T, V) command.



The Trace window becomes active and displays the trace results.

To trace before a particular variable value and break

1. Double-click the desired global variable.
2. Choose the Trace→Variable Break... (ALT, T, B) command.
3. Enter the value in the Value text box.
4. Choose the OK button.

The Trace→Variable Break... (ALT, T, B) command breaks execution as soon as the specified value is written to the specified global variable.

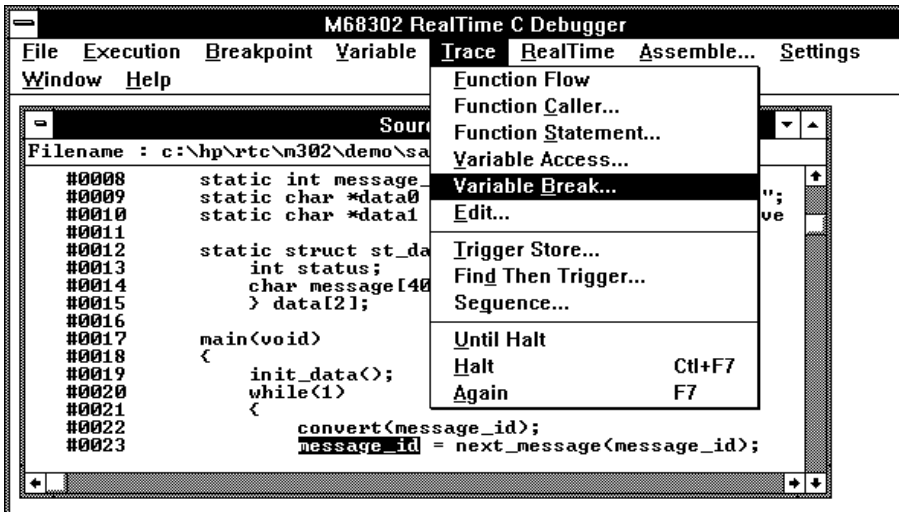
The command also breaks execution at the Assembler symbol specified by its name and size.

Example

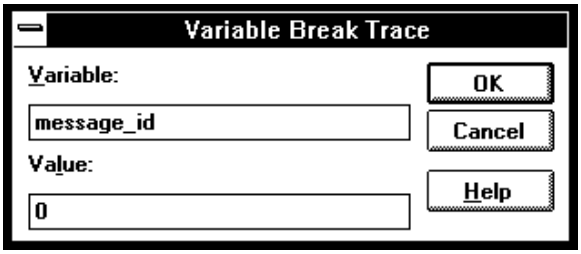
To break execution as soon as "message_id" contains "0":

Double-click "message_id".

Choose the Trace→Variable Break... (ALT, T, B) command.



Enter "0" in the Value text box.



Choose the OK button.

The debugger halts execution as soon as the program writes "0" to the "message_id" variable. Once execution has halted, the Trace window becomes active and displays the results.

To trace until the command is halted

1 To start the trace, choose the Trace→Until Halt (ALT, T, U) command.

2 When you are ready to stop the trace, choose the Trace→Halt (ALT, T, H) command.

This command is useful, for example, in tracing program execution that leads to a processor halted state or to a break to the monitor.

To stop a running trace

- Choose the Trace→Halt (ALT, T, H) command.

The command is used to:

Stop the trace initiated with the Trace→Until Halt (ALT, T, U) command.

Force termination of the trace that cannot be completed due to absence of the specified state.

Stop a trace before the trace buffer becomes full.

To repeat the last trace

- Choose the Trace→Again (ALT, T, A) command, or press the F7 key.

The Trace→Again (ALT, T, A) command traces program execution using the last trace specification stored in the HP 64700.

To identify bus arbitration cycles in the trace

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select the Tag Bus Arbitration for Analyzer option.
3. Choose the OK button to confirm your selection.

When the Tag Bus Arbitration for Analyzer option is selected, a state is stored in trace memory when a bus arbitration cycle occurs.

Bus arbitration tag states can be identified using the dma predefined status value.

To display bus cycles

1. Place the cursor on the line from which you wish to display the bus cycles.
2. From the Trace window's control menu, choose the Display→Mixed Mode (ALT, -, D, M) command or the Display→Bus Cycle Only (ALT, -, D, C) command.

The Display→Mixed Mode (ALT, -, D, M) command displays each source line followed by the bus cycles associated with it.

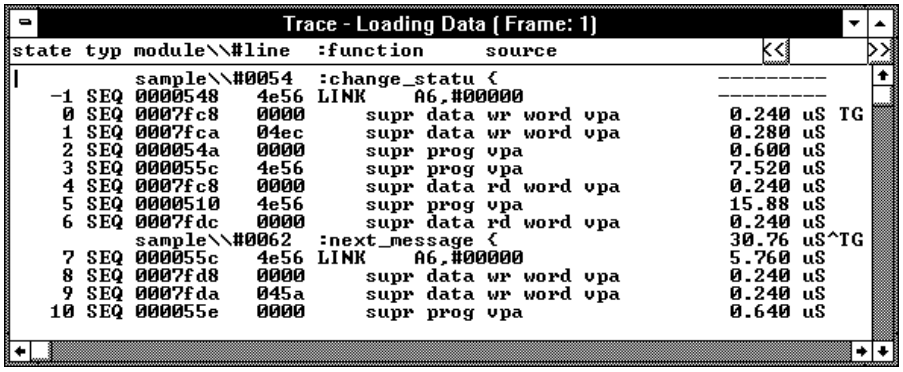
The Display→Bus Cycle Only (ALT, -, D, C) command displays the bus cycles without the source lines.

The display starts from the cursor-selected line.

To hide the bus cycles, choose the Display→Source Only (ALT, -, D, S) command from the Trace window's control menu.

Example

Bus Cycles Displayed in Trace with "Mixed Mode" selected:



To display absolute or relative counts

- From the Trace window's control menu, choose the Display→Count→Absolute (ALT, -, D, C, A) or Display→Count→Relative (ALT, -, D, C, R) command.

  Choosing the Display→Count→Relative (ALT, -, D, C, R) command selects the relative mode where the state-to-state time intervals are displayed.

  Choosing the Display→Count→Absolute (ALT, -, D, C, A) command selects the absolute mode where the trace time is displayed as the total time elapsed since the analyzer has been triggered.

Setting Up Custom Trace Specifications

This section shows you how:

- To set up a "Trigger Store" trace specification
- To set up a "Find Then Trigger" trace specification
- To set up a "Sequence" trace specification
- To edit a trace specification
- To trace "windows" of program execution
- To store the current trace specification
- To load a stored trace specification

Note The analyzer traces unexecuted instructions due to prefetching in 68302.

Note Analyzer memory is unloaded two states at a time. If you use a storage qualifier to capture states that do not occur often, it's possible that one of these states has been captured and stored but cannot be displayed because another state must be stored before the pair can be unloaded. When this happens, you can stop the trace measurement to see all stored states.

When Do I Use the Different Types of Trace Specifications?

When you wish to trigger the analyzer on the occurrence of one state, use the "Trigger Store" dialog box to set up the trace specification.

When you wish to trigger the analyzer on the occurrence of one state followed by another state, or one state followed by another state but only when that state occurs before a third state, use the "Find Then Trigger" dialog box to set up the trace specification.

When you wish to trigger the analyzer on a sequence of more than two states, use the "Sequence" dialog box to set up the trace specification.

To set up a "Trigger Store" trace specification

1 Choose the Trace→Trigger Store... (ALT, T, T) command.

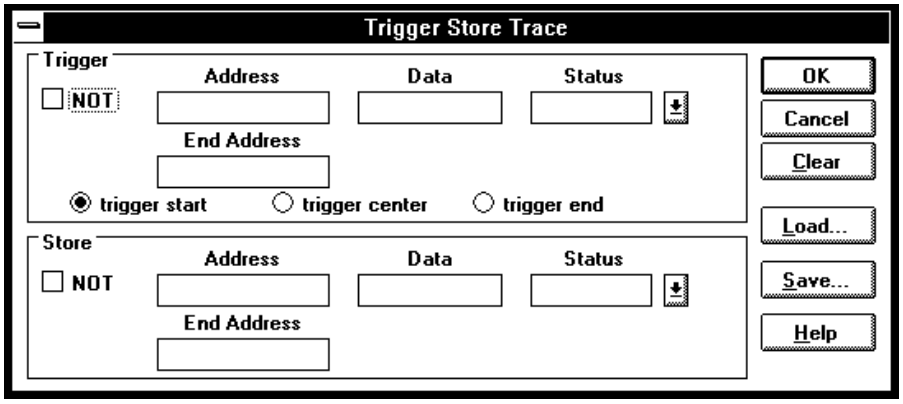
2 Specify the *trigger condition* using the Address, Data, and/or Status text boxes within the Trigger group box.

3 Specify the *trigger position* by selecting the trigger start, trigger center, or trigger end option in the Trigger group box.

4 Specify the *store condition* using the Address, Data, and/or Status text boxes within the Store group box.

5 Choose the OK button to set up the analyzer and start the trace.

The Trace→Trigger Store... (ALT, T, T) command opens the Trigger Store Trace dialog box:



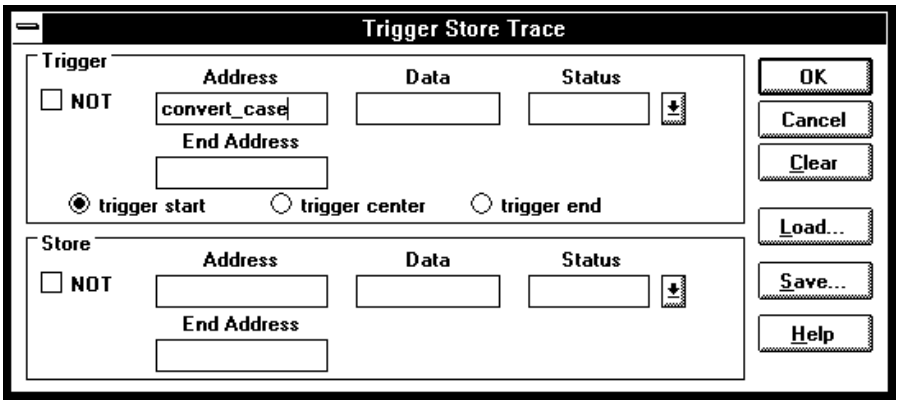
A group of Address, Data, and Status text boxes combine to form a *state qualifier*. You can specify an address range by entering a value in the End Address box. By selecting the NOT check box, you can specify all states other than those identified by the address, data, and *status values*.

Example

To trace execution after the "convert_case" function:

Choose the Trace→Trigger Store... (ALT, T, T) command.

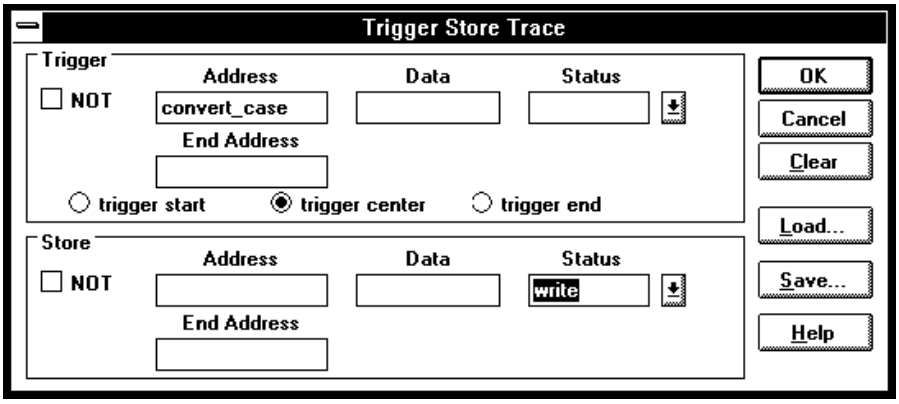
Enter "convert_case" in the Address text box in the Trigger group box.



Choose the OK button.

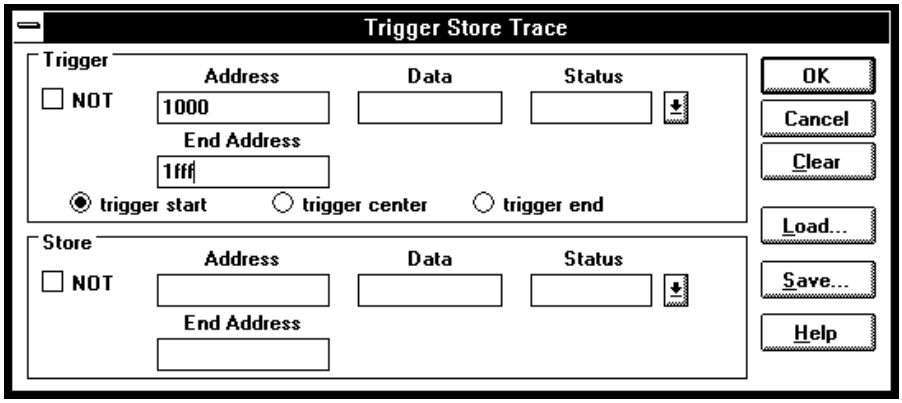
Example

To trace execution before and after the "convert_case" function and store only states with "write" status:



Example

To specify the trigger condition as any address in the range 1000h through 1fffh:



To set up a "Find Then Trigger" trace specification

1 Choose the Trace→Find Then Trigger... (ALT, T, D) command.

2 Specify the sequence, which is made up of the *enable*, *trigger store*, *trigger*, and *store* conditions.

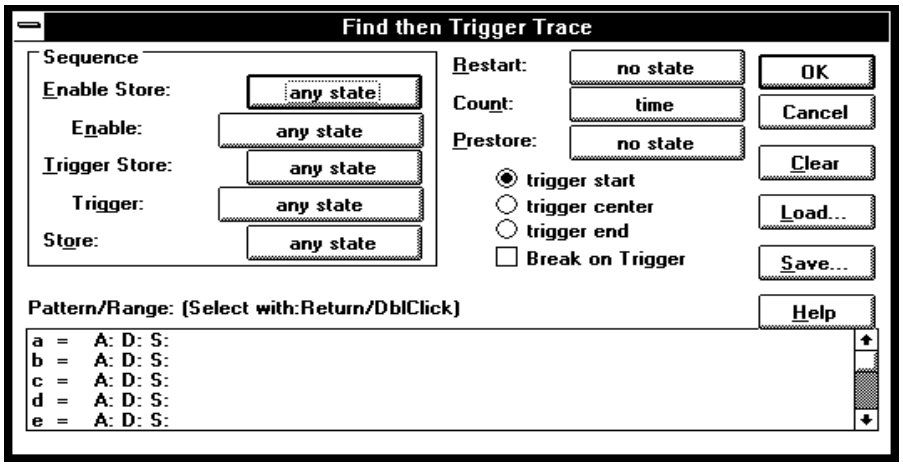
3 Specify the *restart*, *count*, and *prestore* conditions.

4 Specify the *trigger position* by selecting the trigger start, trigger center, or trigger end option.

5 If you want emulator execution to break to the monitor when the trigger condition occurs, select the *Break On Trigger* check box.

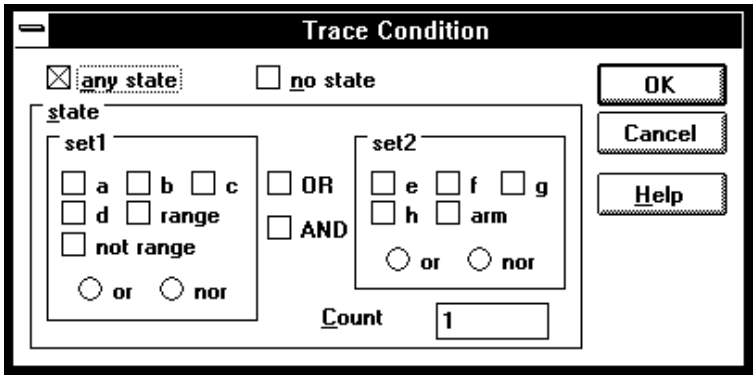
6 Choose the OK button to set up the analyzer and start the trace.

The Trace→Find Then Trigger... (ALT, T, D) command opens the Find Then Trigger Trace dialog box:



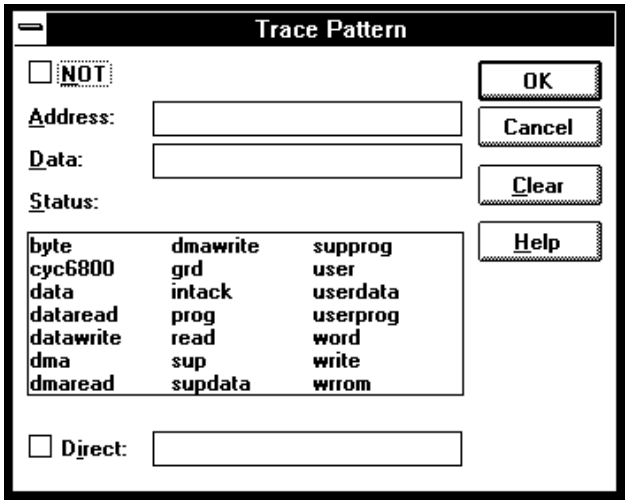
Choosing the enable, trigger, store, count, or prestore buttons opens a Condition dialog box that lets you select "any state", "no state", trace patterns

"a" through "h", "range", or "arm" as the condition. Patterns "a" through "h", "range", and "arm" are grouped into two sets, and resources within a set may be combined using the "or" or "nor" logical operators. Resources from the two sets may be combined using the OR or AND logical operators.

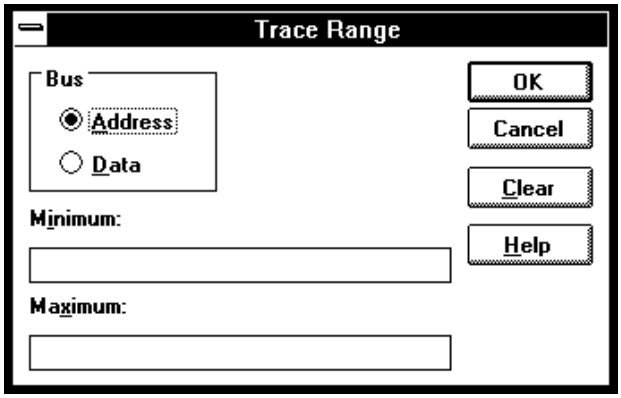


The range and pattern resources are defined by double-clicking on the resource name in the Pattern/Range list box.

If you double-click on a pattern name, the Trace Pattern dialog box is opened to let you specify address, data, and status values. By selecting the NOT check box, you can specify all states other than those identified by the address, data, and *status values*. The Direct check box lets you specify status values other than those that have been predefined.



If you double-click on the range resource, the Trace Range dialog box is opened to let you select either the Address range or the Data range option and enter the minimum and maximum values in the range.



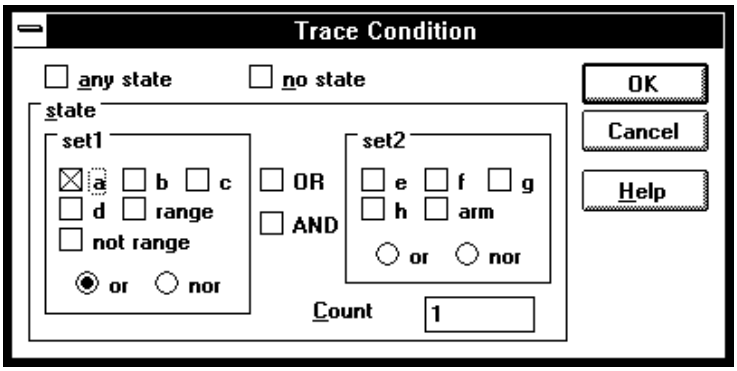
Example

To trace execution after the "convert_case" function:

Choose the Trace→Find Then Trigger... (ALT, T, D) command.

Choose the Trigger button (default: any state).

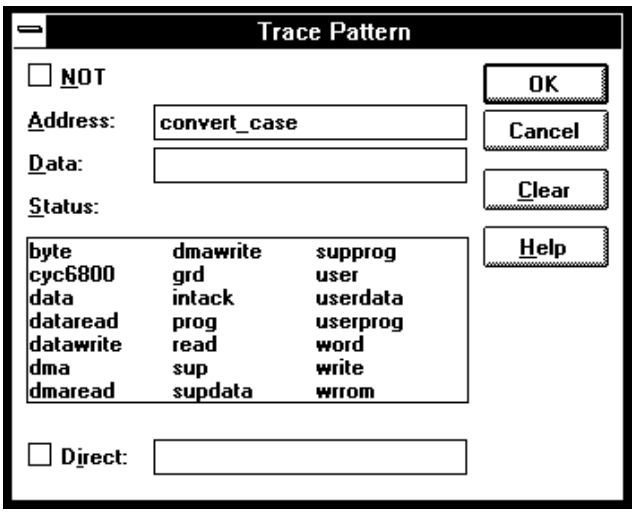
Select "a".



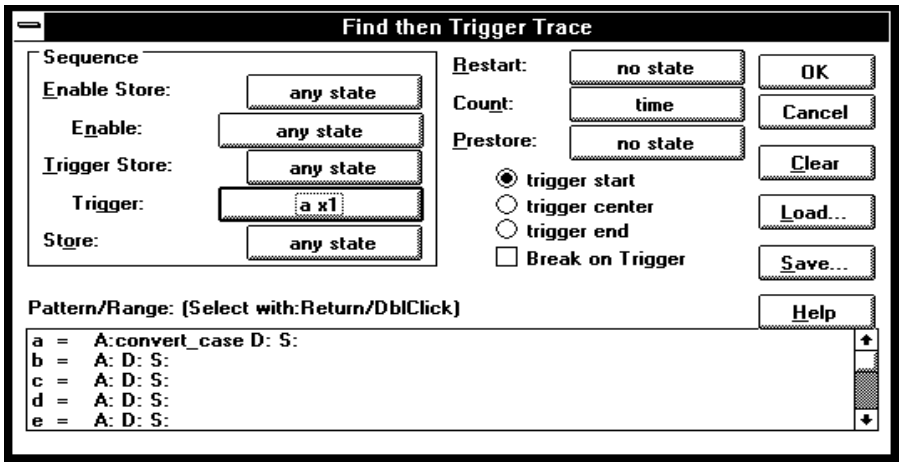
Choose the OK button.

Double-click "a" in the Pattern/Range list box.

Enter "convert_case" in the Address text box in the Trace Pattern dialog box.



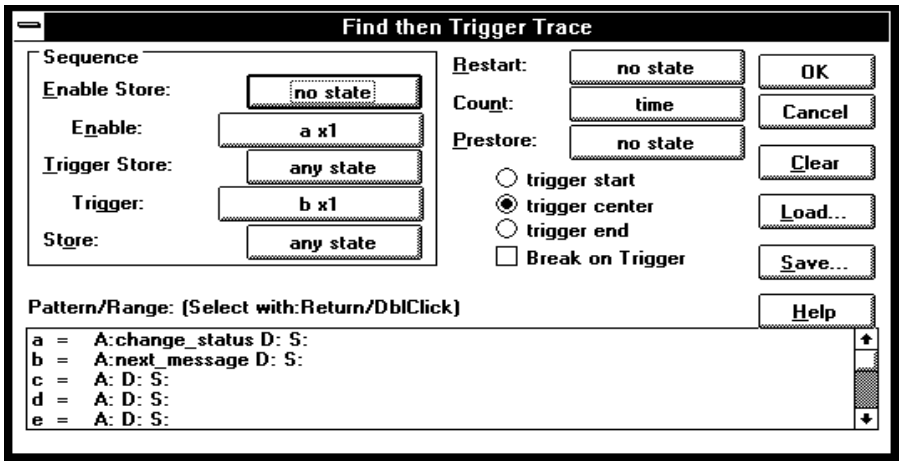
Choose the OK button in the Trace Pattern dialog box.



Choose the OK button in the Find Then Trigger Trace dialog box.

Example

To trace about the "next_message" function when it follows the "change_status" function and store all states after the "change_status" function:



To set up a "Sequence" trace specification

Sequence trace specifications let you trigger the analyzer on a sequence of several captured states.

There are 8 sequence levels. When a trace is started, the first sequence level is active. You select one of the remaining sequence levels as the level that, when entered, will trigger the analyzer. Each level lets you specify two conditions that, when satisfied by a captured state, will cause branches to other levels:

```
if (state matches primary branch condition)
   then GOTO (level associated with primary branch)
else if (state matches secondary branch condition)
   then GOTO (level associated with secondary branch)
else
   stay at current level
```

Note that if a state matches both the primary and secondary branch conditions, the primary branch is taken.

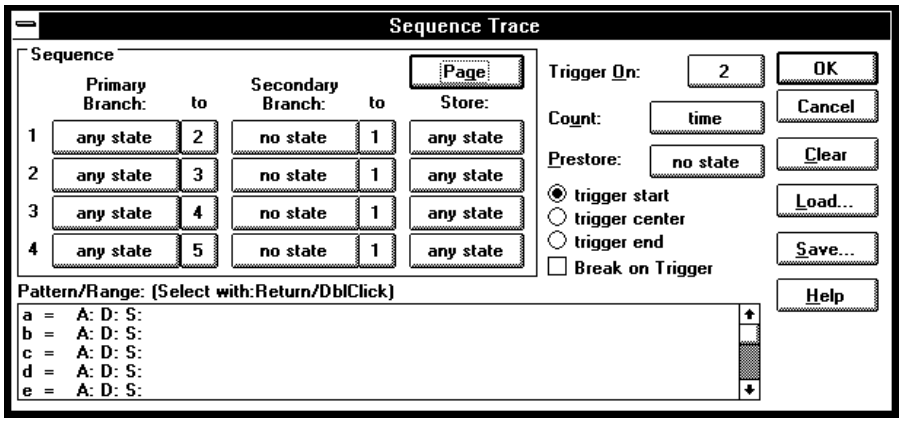
Each sequence level also has a store condition that lets you specify the states that get stored while at that level.

1. Choose the Trace→Sequence... (ALT, T, Q) command.
2. Specify the *primary branch*, *secondary branch*, and *store* conditions for each *sequence level* you will use.
3. Specify which sequence level to trigger on. The analyzer triggers on the entry to the specified level. Therefore, the condition that causes a branch to the specified level actually triggers the analyzer.
4. Specify the *count* and *prestore* conditions.
5. Specify the *trigger position* by selecting the trigger start, trigger center, or trigger end option.

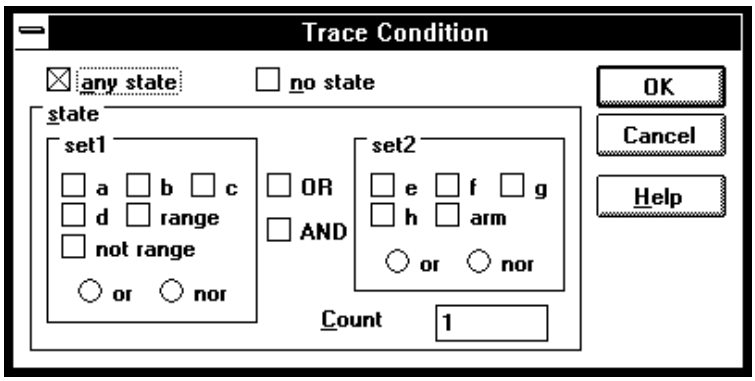
6 If you want emulator execution to break to the monitor when the trigger condition occurs, select the *Break On Trigger* check box.

7 Choose the OK button to set up the analyzer and start the trace.

The Trace→Sequence... (ALT, T, Q) command calls the Sequence Trace Setting dialog box, where you make the following trace specifications:

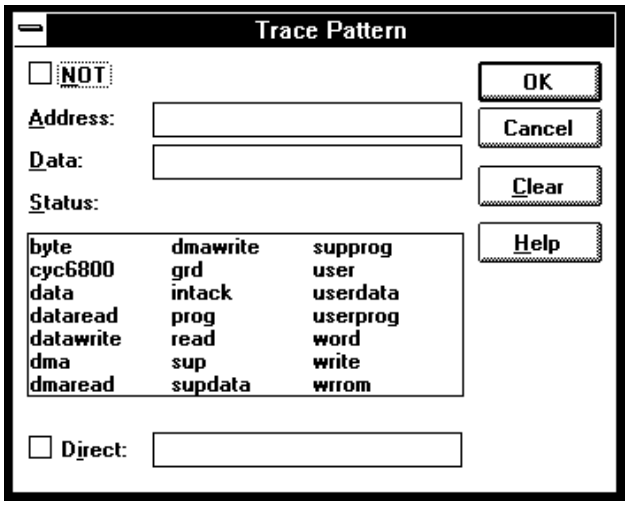


Choosing the primary branch, secondary branch, store, count, or prestore buttons opens a Condition dialog box that lets you select "any state", "no state", trace patterns "a" through "h", "range", or "arm" as the condition. Patterns "a" through "h", "range", and "arm" are grouped into two sets, and resources within a set may be combined using the "or" or "nor" logical operators. Resources in the two sets may be combined using the OR or AND logical operators.

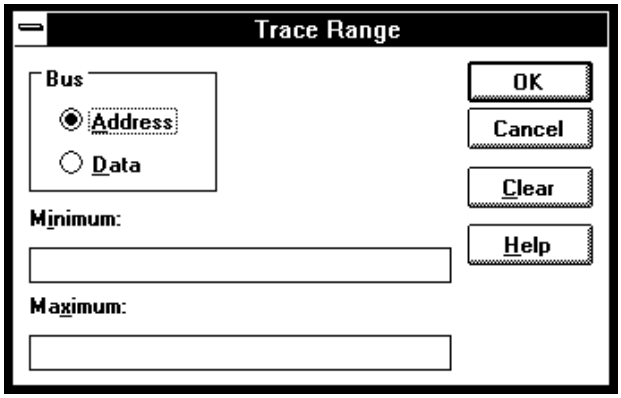


The range and pattern resources are defined by double-clicking on the resource name in the Pattern/Range list box.

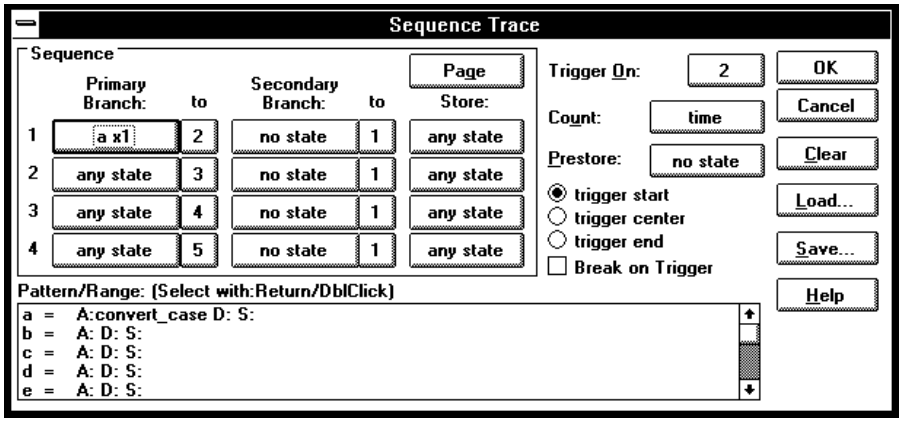
If you double-click on a pattern name, the Trace Pattern dialog box is opened to let you specify address, data, and status values. By selecting the NOT check box, you can specify all states other than those identified by the address, data, and *status values*. The Direct check box lets you specify status values other than those that have been predefined.



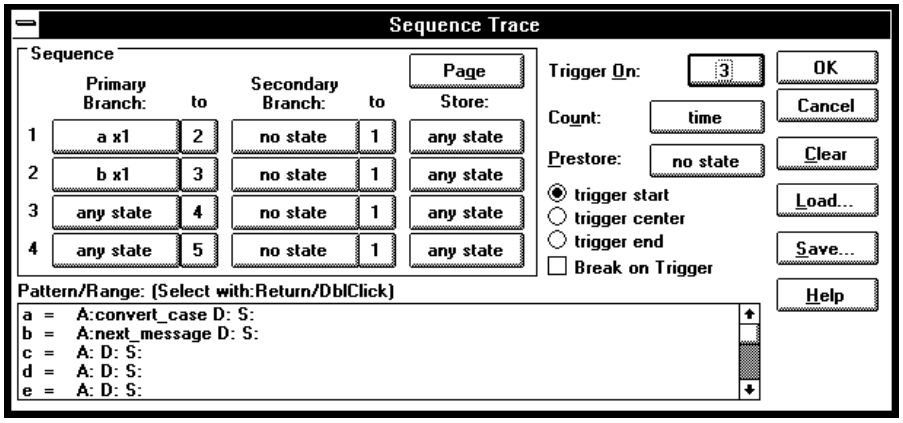
If you double-click on the range resource, the Trace Range dialog box is opened to let you select either the Address range option or the Data range option and enter the minimum and maximum values in the range.



Example To specify address "convert_case" as the trigger condition:



Example To specify execution of "convert_case" and "next_message" as the trigger sequence:



To edit a trace specification

1 Choose the Trace→Edit... (ALT, T, E) command.

2 Using the Sequence Trace dialog box, edit the trace specification as desired.

3 Choose the OK button.

You can use this command to edit trace specifications, including trace specifications that are automatically set up. For example, you can use this command to edit the trace specification that is set up when the Trace→Function Flow (ALT, T, F) command is chosen.

To trace "windows" of program execution

1 Because pairs of sequence levels are used to capture window enable and disable states both before and after the trigger, choose the Trace→Sequence... (ALT, T, Q) command.

2 Set up the sequence levels, patterns, and other trace options (as described below) in the Sequence Trace dialog box.

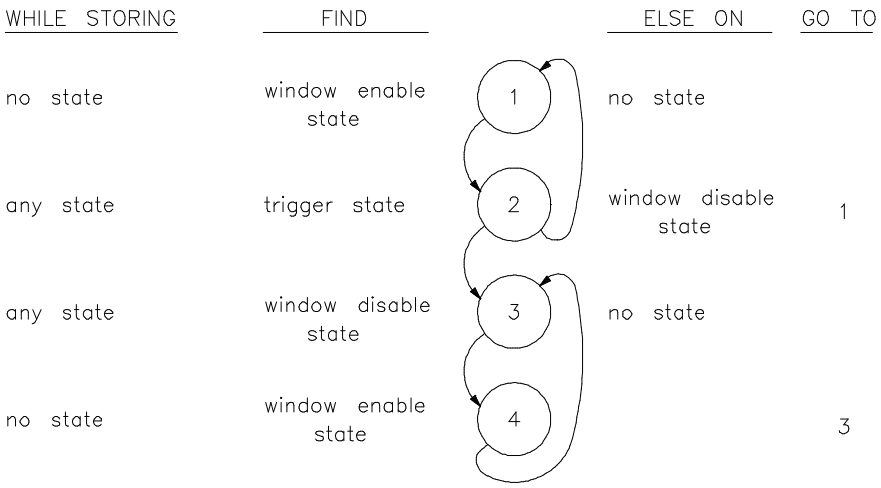
3 Choose the OK button.

When you trace "windows" of program execution, you store states that occur between one state and another state. Storing states that occur between two states is different from the trace specification set up by the Trace→Statement... (ALT, T, S) command, which stores states in a function's range of addresses.

In a typical windowing trace specification, sequence levels are paired. The first sequence level searches for the window enable state, and no states are stored while searching. When the window enable state is found, the second

sequence level stores the states you're interested in while searching for the window disable state.

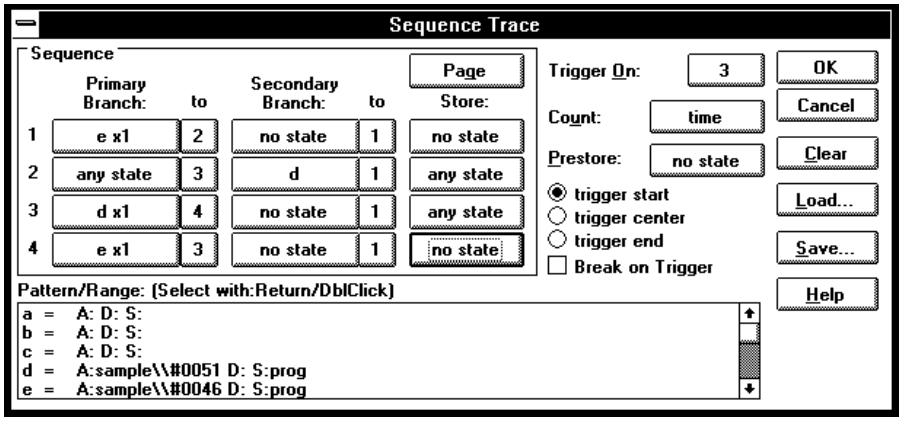
If you want to store the window of code execution before and after the trigger condition, use two sets of paired sequence levels: one window enable/disable pair of sequence levels before the trigger, and another disable/enable pair after the trigger as shown below.



Notice that the order of the second sequence level pair is swapped. In sequence level 2, if the analyzer finds the trigger condition while searching for the window disable state, it will branch to sequence level 3 where it continues its search for the window disable state. After this, the analyzer will remain in sequence levels 3 and 4 until the trace memory is filled, completing the trace.

Example

To trace the window of code execution between lines 46 and 51 of the sample program, triggering on any state in the window:



Notice that the analyzer triggers on the entry to sequence level 3. The primary branch condition in level 2 actually specifies the trigger condition.

To store the current trace specification

1. Choose the Trace→Edit... (ALT, T, E) command.
2. Choose the Save... button.
3. Specify the name of the trace specification file.
4. Choose the OK button.

You can also store trace specifications from the Trigger Store Trace, Find Then Trigger Trace, or Sequence Trace dialog boxes.

The extension for trace specification files defaults to ".TRC".

To load a stored trace specification

1 Choose the Trace→Trigger Store... (ALT, T, T), Trace→Find Then Trigger... (ALT, T, D), Trace→Sequence... (ALT, T, Q), or Trace→Edit... (ALT, T, E) command.

2 Choose the Load... button.

3 Select the desired trace specification file.

4 Choose the OK button.

A "Trigger Store" trace specification file can be loaded into any of the trace setting dialog boxes. A "Find Then Trigger" trace specification file can be loaded into either the Find Then Trigger Trace or Sequence Trace dialog boxes. A "Sequence" trace specification file can only be loaded into the Sequence Trace dialog box.

Part 3

Reference

Descriptions of the product in a dictionary or encyclopedia format.

6

Command File and Macro Command Summary

Command File and Macro Command Summary

This section lists the Real-Time C Debugger break macro and command file commands, providing syntax and brief description for each of the listed commands. For details on each command, refer to the command descriptions.

The characters in parentheses can be ignored for shortcut entry.

Run Control Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
BRE(AK)					Breaking execution
COM(E)					Run to cursor-indicated line
OVE(R)					Stepping over
OVE(R)	count				Repeated a number of times
OVE(R)	count	address			From specified address
OVE(R)	count	STA(RT)			From transfer address
RES(ET)					Resetting processor
RET(URN)					Until return
RUN					From current address
RUN	address				From specified address
RUN	STA(RT)				From transfer address
RUN	RES(ET)				From reset
STE(P)					Stepping
STE(P)	count				Repeated a number of times
STE(P)	count	address			From specified address
STE(P)	count	STA(RT)			From transfer address

Variable and Memory Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
MEM(ORY)	address				Changing address displayed
MEM(ORY)	address	TO	value		Edit memory, display size
MEM(ORY)	size	address	TO	value	Edit memory, specify size
MEM(ORY)	FIL(L)	size	addr-range	value	Filling memory contents
MEM(ORY)	COP(Y)	size	addr-range	address	Copying memory contents
MEM(ORY)	IMA(GE)	size	addr-range		Copying target memory
MEM(ORY)	LOA(D)	format	filename		Loading memory from a file
MEM(ORY)	STO(RE)	format	addr-range	filename	Storing memory to a file
MEM(ORY)	BYT(E)				Byte format display
MEM(ORY)	WOR(D)				16-Bit format display
MEM(ORY)	ABS(OLUTE)				Single-column display
MEM(ORY)	BLO(CK)				Multi-column display
MEM(ORY)	LON(G)				32-Bit format display
IO	SET	size	space	address	Registering I/O display
IO	DEL(ETE)	size	space	address	Deleting I/O display
IO	size	space	address	TO value	Editing I/O
VAR(IABLE)	address	TO	value		Editing variable

```
WP          SET         address                          Registering watchpoint
WP          DEL(ETE)    address                          Deleting watchpoint
WP          DEL(ETE)    ALL                              Deleting all watchpoints
```

Breakpoint Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
MODE	BKP(TBREAK)	ON|OFF			Deletes all/prevents new breakpoints
BM	SET	linenumber	command		Setting break macro
BM	SET	plinenum	command		Setting break macro
BM	DEL(ETE)	linenumber			Deleting break macro
BM	DEL(ETE)	plinenum			Deleting break macro
BP	SET	address			Setting breakpoint
BP	DEL(ETE)	address			Deleting breakpoint
BP	DEL(ETE)	ALL			Deleting breakpoint
BP	DISABLE	address			Disabling a breakpoint
BP	ENABLE	address			Enabling a breakpoint
EVA(LUATE)	address				Expression window display
EVA(LUATE)	"strings"				Printing string
EVA(LUATE)	CLE(AR)				Clearing Expression window

Window Open/Close Command

Command	Param_1	Param_2	Param_3	Param_4	Operation
DIS(PLAY)	window-name				Opening the named window
ICO(NIC)	window-name				Closing the named window

Configuration Command

Command	Param_1	Param_2	Param_3	Param_4	Operation
MON(ITOR)	STA(RT)				Starting monitor
MON(ITOR)	mon-item	mon-ans			Setting up monitor
MON(ITOR)	END				Ending monitor
CON(FIG)	STA(RT)				Starting configuration
CON(FIG)	config-item	config-ans			Executing configuration
CON(FIG)	END				Ending configuration
MAP	STA(RT)				Starting mapping
MAP	addr-range	memtype	func-code		Executing mapping
MAP	OTHER	memtype	func-code		Mapping OTHER area
MAP	END				Ending mapping
MOD(E)	MNE(MONIC)	ON			Enabling Mnemonic display
MOD(E)	MNE(MONIC)	OFF			Enabling Source display
MOD(E)	REA(LTIME)	ON			Enabling real-time mode
MOD(E)	REA(LTIME)	OFF			Disabling real-time mode
MOD(E)	IOG(UARD)	ON			Enabling I/O guard
MOD(E)	IOG(UARD)	OFF			Disabling I/O guard
MOD(E)	MEM(ORYPOLL)	ON			Enabling Memory polling
MOD(E)	MEM(ORYPOLL)	OFF			Disabling Memory polling
MOD(E)	WAT(CHPOLL)	ON			Enabling WatchPoint polling
MOD(E)	WAT(CHPOLL)	OFF			Disabling WatchPoint polling
MOD(E)	LOG	ON			Enabling log file output
MOD(E)	LOG	OFF			Disabling log file output
MOD(E)	BNC	IN			Setting BNC input
MOD(E)	BNC	OUT			Setting BNC output
MOD(E)	SYM(BOLCASE)	ON			Case sensitive symbol search
MOD(E)	SYM(BOLCASE)	OFF			Case insensitive sym. search
MOD(E)	DOW(NLOAD)	ERR(ABORT)			Error causes load abort
MOD(E)	DOW(NLOAD)	NOE(RRABORT)			Load continues after error
MOD(E)	SOU(RCE)	ASK(PATH)			Prompt for source paths
MOD(E)	SOU(RCE)	NOA(SKPATH)			Don’t prompt for src paths

MOD(E)	TRACECLOCK	BACKGROUND			Trace background cycles
MOD(E)	TRACECLOCK	BOTH			Trace all processor cycles
MOD(E)	TRACECLOCK	USER			Trace user program cycles

File Command

Command	Param_1	Param_2	Param_3	Param_4	Operation
FIL(E)	SOU(RCE)	modulename			Displaying source file
FIL(E)	OBJ(ECT)	filename	func-code		Loading object
FIL(E)	SYM(BOL)	filename	func-code		Loading symbol
FIL(E)	BIN(ARY)	filename	func-code		Loading data
FIL(E)	APPEND	filename	func-code		Appending symbol
FIL(E)	CHA(INCMD)	filename			Chaining command files
FIL(E)	COM(MAND)	filename			Executing command file
FIL(E)	LOG	filename			Specifying command log file
FIL(E)	RER(UN)				Re-executes command file
FIL(E)	CON(FIGURATION)	LOA(D)	filename		Loads config. from file
FIL(E)	CON(FIGURATION)	STO(RE)	filename		Stores configuration to file
FIL(E)	ENV(IRONMENT)	LOA(D)	filename		Loads environment from file
FIL(E)	ENV(IRONMENT)	SAV(E)	filename		Stores environment to file

Trace Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
TRA(CE)	FUN(CTION)	FLO(W)			Tracing function flow
TRA(CE)	FUN(CTION)	CAL(L)	funcname		Tracing function call
TRA(CE)	FUN(CTION)	STA(TEMENT)	funcname		Tracing statement
TRA(CE)	VAR(IABLE)	ACC(ESS)	address		Tracing access to variable
TRA(CE)	VAR(IABLE)	BRE(AK)	address	value	Setting breakpoint variable
TRA(CE)	STO(P)				Stopping tracing
TRA(CE)	ALW(AYS)				Tracing until halt
TRA(CE)	AGA(IN)				Restarting tracing
TRA(CE)	SAV(E)	filename			Storing trace specification
TRA(CE)	LOA(D)	filename			Loading trace specification
TRA(CE)	CUS(TOMIZE)				Starts trace w/loaded spec.
TRA(CE)	DIS(PLAY)	MIX(ED)			Enabling source+bus display
TRA(CE)	DIS(PLAY)	SOU(RCE)			Enabling source display
TRA(CE)	DIS(PLAY)	BUS			Enabling bus display
TRA(CE)	DIS(PLAY)	ABS(OLUTE)			Displaying absolute time
TRA(CE)	DIS(PLAY)	REL(ATIVE)			Displaying relative time
TRA(CE)	COP(Y)	DISPLAY			Copying trace display
TRA(CE)	COP(Y)	ALL			Copying trace results
TRA(CE)	FIN(D)	TRI(GGER)			Centers trigger in window
TRA(CE)	FIN(D)	STA(TE)	state-num		Centers state in window
TRA(CE)	COP(Y)	SPE(C)			Copying specification

Symbol Window Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
SYM(BOL)	LIS(T)	MOD(ULE)			Displaying module
SYM(BOL)	LIS(T)	FUN(CTION)			Displaying function
SYM(BOL)	LIS(T)	EXT(ERNAL)			Displaying global symbol
SYM(BOL)	LIS(T)	INT(ERNAL)	funcname		Displaying local symbol
SYM(BOL)	LIS(T)	GLO(BAL)			Displaying global asm symbol
SYM(BOL)	LIS(T)	LOC(AL)	modulename		Displaying local asm symbol
SYM(BOL)	ADD	usersymbol	address		Adding user-defined symbol
SYM(BOL)	DEL(ETE)	usersymbol			Deleting user-defined symbol
SYM(BOL)	DEL(ETE)	ALL			Deleting all user symbols
SYM(BOL)	MAT(CH)	"strings"			Displaying matched string
SYM(BOL)	COP(Y)	DIS(PLAY)			Copying symbol display
SYM(BOL)	COP(Y)	ALL			Copying all symbols

Command File Control Command

Command	Param_1	Param_2	Param_3	Param_4	Operation
EXIT					Exiting command file
EXIT	VAR(IABLE)	address	value		Exiting with variable cont.
EXIT	REG(ISTER)	regname	value		Exiting with register cont.
EXIT	MEM(ORY)	size	address	value	Exiting with memory contents
EXIT	IO	BYTE/WORD	address	value	Exiting with I/O contents
WAIT	MON(ITOR)				Wait until MONITOR status
WAIT	RUN				Wait until RUN status
WAIT	UNK(NOWN)				Wait until UNKNOWN status
WAIT	SLO(W)				Wait until SLOW CLOCK status
WAIT	TGT(RESET)				Wait until TARGET RESET
WAIT	SLE(EP)				Wait until SLEEP status
WAIT	GRA(NT)				Wait until BUS GRANT status
WAIT	NOB(US)				Wait until NOBUS status
WAIT	TCO(M)				Wait until end of trace
WAIT	THA(LT)				Wait until halt
WAIT	TIM(E)	seconds			Wait a number of seconds

Miscellaneous Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
ASM	address	usersymbol	"inst-string"		In-line assembler
BEE(P)					Sounding beep
BUTTON	label	"command"			Adds button to Button window
QUI(T)					Exiting debugger
COP(Y)	TO	filename			Specifying copy destination
COP(Y)	SOU(RCE)				Copying Source window
COP(Y)	REG(ISTER)				Copying Register window
COP(Y)	MEM(ORY)				Copying Memory window
COP(Y)	WAT(CHPOINT)				Copying WatchPoint window
COP(Y)	BAC(KTRACE)				Copying BackTrace window
COP(Y)	IO				Copying I/O window
COP(Y)	EXP(RESSION)				Copying Expression window
COV	RES(ET)				Resetting coverage memory
COV	ON/OFF				Displaying coverage
CUR(SOR)	address				Positioning cursor
CUR(SOR)	PC				Finding current PC
DIR(ECTORY)	directoryname				Directory for source search
NOP					Non-operative
REG(ISTER)	regname	TO	value		Editing register contents
SEA(RCH)	STR(ING)	direction	case	strings	Searching string
SEA(RCH)	FUN(CTION)	funcname			Selecting function
SEA(RCH)	MEM(ORY)	size	addr-range	value	Searching memory
SEA(RCH)	MEM(ORY)	STR(ING)	"strings"		Searching memory for string
TER(MCOM)	ti-command				Terminal Interface command

Parameters

Parameter	Description	Notation
address	Address	See "Reference".
addr-range	Address range	
case	Case sensing	
command	Macro command	Commands listed in the "Reference".
config-ans	Setting	See "Reference".
config-item	Configuration	See "Reference".
count	Count	Decimal notation
direction	Search direction	
directoryname	Directory name	
filename	File name	
format	Memory file format	
funcname	Function name	
func-code	Function code	
label	Button label	
linenumber	Line number	
memtype	Memory type	
modulename	Module name	
mon-ans	Setting	See "Reference".
mon-item	Configuration	See "Reference".
plinenum	Macro line number	line number.macro number (ex. 34.1)
regname	Register name	
seconds	Time in seconds	
size	Data size	
space	Memory or I/O space	
strings	String	"string"
usersymbol	User-defined symbol	See "Reference".
value	Value	See "Reference".
window-name	Window name (1st 3 characters)	See "Reference."

WAIT Command Dialog Box

This dialog box appears when the WAIT command is included in a command file, break macro, or button.

Choosing the STOP button cancels the WAIT command.

7

Expressions in Commands

Expressions in Commands

When you enter values and addresses in commands, you can use:

- Numeric constants (hexadecimal, decimal, octal, or binary values).
- Symbols (identifiers).
- Function codes.
- C operators (pointers, arrays, structures, unions, unary minus operators) and parentheses (specifying the order of operator evaluation).

Numeric Constants

All numeric constants are assumed to be hexadecimal, except when the number refers to a count; count values are assumed to be decimal. By appending a suffix to the numeric value, you can specify its base.

The debugger expressions support the following numeric constants with or without radix:

Hexadecimal	Alphanumeric strings starting with "0x" or "0X" and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 0x12345678, 0xFFFF0000). Alphanumeric strings starting with any of '0' through '9', ending with 'H' or 'h', and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 12345678H, 0FFFF0000h). Alphanumeric strings starting with any of '0' through '9' and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 12345678, 0FFFF0000). Hexadecimal strings starting with alphabetical characters must be preceded by 0. For example, FF40H must be entered as 0FF40H.
Decimal	Numeric strings consisting of any of '0' through '9' and ending with 'T' or 't' (for example: 128T, 1000t).
Octal	Numeric strings consisting of any of '0' through '7' and ending with 'O' or 'o' (not zero) (for example: 200o, 377O).
Binary	Numeric strings consisting of '0' or '1' and ending with 'Y' or 'y' (for example: 10000000y, 11001011Y).
Don't Care	Numeric strings containing 'X' or 'x' values. All numeric strings must begin with a numeric value. For example, x1x0y must be entered as 0x1x0y.

Symbols

The debugger expressions support the following symbols (identifiers):

- Symbols defined in C source code.
- Symbols defined in assembly language source code.
- Symbols added with the Symbol window control menu's User defined→Add... (ALT, -, U, A) command.
- Line number symbols.

Symbol expressions may be in the following format (where bracketed parts are optional):

```
[module_name\\]symbol_name[,format_spec]
```

Module Name

The module names include C/Assembler module names as follows:

Assembler module name	(file_path)asm_file_name
C module name	source_file_name (without extension)

Symbol Name

The symbol names include symbols defined in C/Assembler source codes, user-defined symbols, and line number symbols:

User-defined symbols	Strings consisting of up to 256 characters including: alphanumeric characters, _ (underscore), and ? (question mark).
Line number symbols	#source_file_line_number

The symbol names can also include either * or & to explicitly specify the evaluation of the symbol.

Symbol address	&symbol_name
Symbol data	*symbol_name

Format Specification

The format specifications define the variable display format or size for the variable access or break tracing:

String	s
Decimal	d (current size), d8 (8 bit), d16 (16 bit), d32 (32 bit)
Unsigned decimal	u (current size), u8 (8 bit), u16 (16 bit), u32 (32 bit)
Hexadecimal	x (current size), x8 (8 bit), x16 (16 bit), x32 (32 bit)

Examples

Some example symbol expressions are shown below:

```
sample\\#22,x32
```

Display the address of line number 22 in the module "sample," formatted as a 32-bit hex number. This form (with the format specification) is used in the watchpoint window, expression window, etc.

```
sample\\#22
```

Refer to the address of line number 22 in the module "sample." This form (without the format specification) is used in the trace specification, memory display window, etc.

```
data[2].message,s
```

Display the structure element "message" in the third element of the array "data" as a string.

```
dat→message,s
```

Display the structure element "message" pointed to by the "dat" pointer as a string.

```
dat→message,x32
```

Display the structure element "message" pointed to by the "dat" pointer as a 32-bit hex number.

```
sample\\data[1].status,d32
```

Display the structure element "status" in the second element of the array "data" that is in the module "sample" as a 32-bit decimal integer.

```
&data[0]
```

Refer to the address of the first element of the array "data."

```
*1000
```

Does not do anything. (It displays dashes, as an indication of a parsing error.) Note that you cannot use constants as an address.

Function Codes

Addresses can be specified with any of the *function codes*. The function codes are appended to the addresses, preceded by @ (for example: 0a3bc@up).

You must include a function code when referring to an address that was mapped with a function code other than X. This general rule is true except when:

- Specifying addresses in trace commands (because address qualifiers are compared with values captured on the address bus -- function code information is captured as part of the bus cycle status).
- Referring to a program counter address (because the function code is determined by the Supervisor/User status flag bit).

C Operators

The debugger expressions support the following C operators. The order of operator evaluation can be modified using parentheses '(' and ')'; however, it basically follows C conventions:

Pointers	'*' and '&'
Arrays	'[' and ']'
Structures or unions	'.' and "->"
Unary minus	'-'

8

Menu Bar Commands

Menu Bar Commands

This chapter describes the commands that can be chosen from the menu bar. Command descriptions are in the order they appear in the menu bar (top to bottom, left to right).

- File→Load Object... (ALT, F, L)
- File→Command Log→Log File Name... (ALT, F, C, N)
- File→Command Log→Logging ON (ALT, F, C, O)
- File→Command Log→Logging OFF (ALT, F, C, F)
- File→Run Cmd File... (ALT, F, R)
- File→Load Debug... (ALT, F, D)
- File→Save Debug... (ALT, F, S)
- File→Load Emulator Config... (ALT, F, E)
- File→Save Emulator Config... (ALT, F, V)
- File→Copy Destination... (ALT, F, P)
- File→Exit (ALT, F, X)
- File→Exit HW Locked (ALT, F, H)
- Execution→Run (ALT, E, U)
- Execution→Run to Cursor (ALT, R C)
- Execution→Run to Caller (ALT, E, T)
- Execution→Run... (ALT, E, R)
- Execution→Single Step (ALT, E, N)
- Execution→Step Over (ALT, E, O)
- Execution→Step... (ALT, E, S)
- Execution→Break (ALT, E, B)
- Execution→Reset (ALT, E, E)
- Breakpoint→Set at Cursor (ALT, B, S)
- Breakpoint→Delete at Cursor (ALT, B, D)
- Breakpoint→Set Macro... (ALT, B, M)
- Breakpoint→Delete Macro (ALT, B, L)
- Breakpoint→Edit... (ALT, B, E)
- Variable→Edit... (ALT, V, E)
- Trace→Function Flow (ALT, T, F)
- Trace→Function Caller... (ALT, T, C)
- Trace→Function Statement... (ALT, T, S)

- Trace→Variable Access... (ALT, T, V)
- Trace→Variable Break... (ALT, T, B)
- Trace→Edit... (ALT, T, E)
- Trace→Trigger Store... (ALT, T, T)
- Trace→Find Then Trigger... (ALT, T, D)
- Trace→Sequence... (ALT, T, Q)
- Trace→Until Halt (ALT, T, U)
- Trace→Halt (ALT, T, H)
- Trace→Again (ALT, T, A)
- RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D)
- RealTime→Monitor Intrusion→Allowed (ALT, R, T, A)
- RealTime→I/O Polling→ON (ALT, R, I, O)
- RealTime→I/O Polling→OFF (ALT, R, I, F)
- RealTime→Watchpoint Polling→ON (ALT, R, W, O)
- RealTime→Watchpoint Polling→OFF (ALT, R, W, F)
- RealTime→Memory Polling→ON (ALT, R, M, O)
- RealTime→Memory Polling→OFF (ALT, R, M, F)
- Assemble... (ALT, A)
- Settings→Emulator Config→Hardware... (ALT, S, E, H)
- Settings→Emulator Config→Memory Map... (ALT, S, E, M)
- Settings→Emulator Config→Monitor... (ALT, S, E, O)
- Settings→Communication... (ALT, S, C)
- Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O)
- Settings→BNC→Input to Analyzer Arm (ALT, S, B, I)
- Settings→Coverage→Coverage ON (ALT, S, V, O)
- Settings→Coverage→Coverage OFF (ALT, S, V, F)
- Settings→Coverage→Coverage Reset (ALT, S, V, R)
- Settings→Font... (ALT, S, F)
- Settings→Tabstops... (ALT, S, T)
- Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O)
- Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F)
- Settings→Extended→Trace Cycles→User (ALT, S, X, T, U)
- Settings→Extended→Trace Cycles→Monitor (ALT, S, X, T, M)
- Settings→Extended→Trace Cycles→Both (ALT, S, X, T, B)
- Settings→Extended→Load Error Abort→ON (ALT, S, X, L, O)
- Settings→Extended→Load Error Abort→OFF (ALT, S, X, L, F)
- Settings→Extended→Source Path Query→ON (ALT, S, X, S, O)
- Settings→Extended→Source Path Query→OFF (ALT, S, X, S, F)

- Window→Cascade (ALT, W, C)
- Window→Tile (ALT, W, T)
- Window→Arrange Icons (ALT, W, A)
- Window→1-9 <win_name> (ALT, W, 1-9)
- Window→More Windows... (ALT, W, M)
- Help→About Debugger/Emulator... (ALT, H, D)

File→Load Object... (ALT, F, L)

Loads the specified object file and symbolic information into the debugger.

Program code is loaded into emulation memory or target system RAM.

Object files must be IEEE-695 format absolute files. Some software development tools that generate this format are:

Microtec MCC68K Compiler

Microtec ASM68K Assembler

Microtec LNK68K Linker

HP AxLS CC68000 Compiler

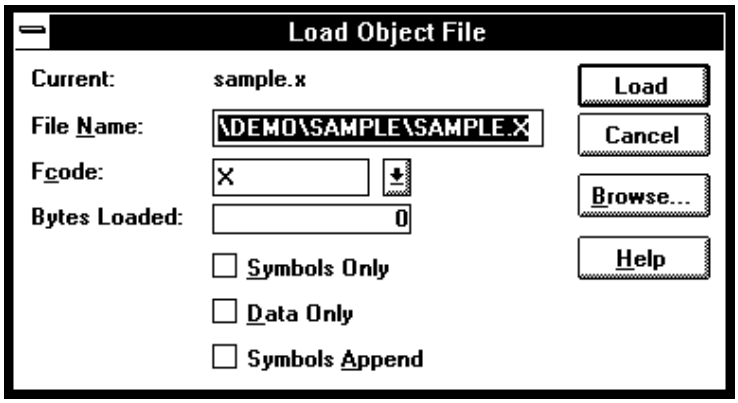
HP AxLS AS68K Assembler

HP AxLS LD68K Linker

You can also load Motorola S-Record and Intel Hexadecimal format files; however, no symbolic information from these files will be loaded.

Load Object File Dialog Box

Choosing the File→Load Object... (ALT, F, L) command opens the following dialog box:



Current	Shows the currently loaded object file.
File Name	Specifies the object file to be loaded. The system defaults the file extension to ".x".
Fcode	Assigns any of the *function codes* to the destination memory area.
Bytes Loaded	Displays the loaded data in Kbytes.
Symbols Only	Loads only the symbolic information. This is used when programs are already in memory (for example, when the debugger is exited and reentered without turning OFF power to the target system or when code is in target system ROM).
Data Only	Loads program code but not symbols.
Symbols Append	Appends the symbols from the specified object file to the currently loaded symbols. This lets you debug code loaded from multiple object files.
Load	Starts loading the specified object file and closes the dialog box.
Cancel	Closes the dialog box without loading the object file.
Browse...	Opens a file selection dialog box from which you can select the object file to be loaded.

Command File Command

`FIL(E) OBJ(ECT) file_name func_code`
Loads the specified object file and symbols into the debugger.

`FIL(E) SYM(BOL) file_name func_code`
Loads only the symbolic information from the specified object file.

`FIL(E) BIN(ARY) file_name func_code`
Loads only the program code from the specified object file.

`FIL(E) APP(END) file_name func_code`
Appends the symbol information from the specified object file to the currently loaded symbol information.

See Also

"To load user programs" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

File→Command Log→Log File Name... (ALT, F, C, N)

Lets you name a new command log file.

The current command log file is closed and the specified command log file is opened. The default command log file name is "log.cmd".

Command log files can be executed with the File→Run Cmd File... (ALT, F, R) command.

The File→Command Log→Logging OFF (ALT, F, C, F) command stops the logging of executed commands.

This command opens a file selection dialog box from which you can select the command log file. Command log files have a ".CMD" extension.

Command File Command

```
FIL(E) LOG filename
```

See Also

"To create a command file" in the "Using Command Files" section of the "Using the Debugger Interface" chapter.

File→Command Log→Logging ON (ALT, F, C, O)

Starts command log file output.

The File→Command Log→Log File Name... (ALT, F, C, N) command specifies the destination file.

Command File Command

```
MOD(E) LOG ON
```

See Also

"To create a command file" in the "Using Command Files" section of the "Using the Debugger Interface" chapter.

File→Command Log→Logging OFF (ALT, F, C, F)

Stops command log file output.

The File→Command Log→Log File Name... (ALT, F, C, N) command specifies the destination file.

Command File Command

```
MOD(E) LOG OFF
```

See Also

"To create a command file" in the "Using Command Files" section of the "Using the Debugger Interface" chapter.

File→Run Cmd File... (ALT, F, R)

Executes the specified command file.

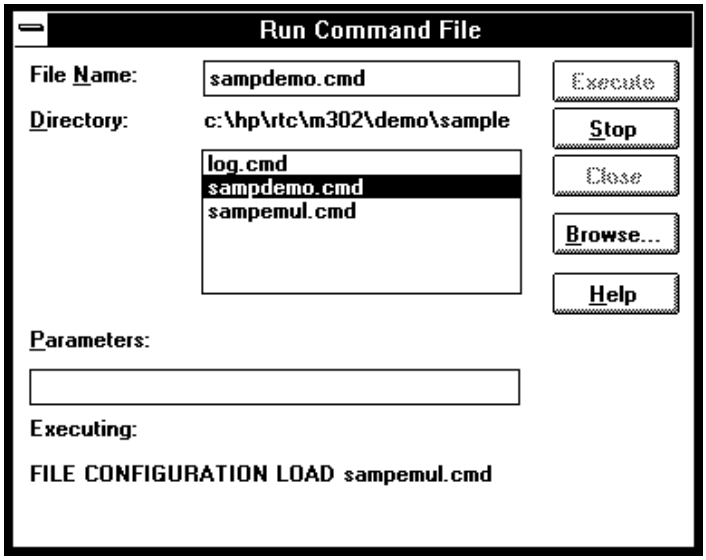
Command files can be:

- Files created with the File→Command Log→Log File Name... (ALT, F, C, N) command.
- Configuration files having .CMD extension.

Command files are stored as ASCII text files so they can be created or edited with ASCII text editors.

Command File Execution Dialog Box

Choosing the File→Run Cmd File... (ALT, F, R) command opens the following dialog box:



File Name	Lets you enter the name of the command file to be executed.
Directory	Shows the current directory and the command files in that directory. You can select the command file name from this list.
Parameters	Lets you specify up to five parameters that replace placeholders $1 through $5 in the command file. Parameters must be separated by blank spaces.
Executing	Shows the command being executed.
Execute	Executes the command file.
Stop	Stops command file execution.
Close	Closes the dialog box.
Browse...	Opens a file selection dialog box from which you can select the command file name.

Command File Command

```
FIL(E) COM(MAND) filename args
```

See Also

"To execute a command file" in the "Using Command Files" section of the "Using the Debugger Interface" chapter.

File→Load Debug... (ALT, F, D)

Loads a debug environment file.

This command opens a file selection dialog box from which you select the debug environment file.

Debug environment files have the extension ".ENV".

Debug environment files contain information about:

- Breakpoints.
- Variables in the WatchPoint window.
- The directory that contains the currently loaded object file.

Command File Command

```
FIL(E) ENV(IRONMENT) LOA(D) filename
```

File→Save Debug... (ALT, F, S)

Saves a debug environment file.

This command opens a file selection dialog box from which you select the debug environment file.

The following information is saved in the debug environment file:

- Breakpoints.
- Variables in the WatchPoint window.
- The directory that contains the currently loaded object file.

Command File Command

```
FIL(E) ENV(IRONMENT) SAV(E) filename
```

File→Load Emulator Config... (ALT, F, E)

Loads a hardware configuration command file.

This command opens a file selection dialog box from which you select the hardware configuration file.

Emulator configuration command files contain:

- Hardware configuration settings.
- Memory map configuration settings.
- Monitor configuration settings.

Command File Command

```
FIL(E) CON(FIGURATION) LOA(D) filename
```

See Also

"To load an emulator configuration" in the "Saving and Loading Configurations" section of the "Configuring the Emulator" chapter.

File→Save Emulator Config... (ALT, F, V)

Saves the current hardware configuration to a command file.

The following information is saved in the emulator configuration file:

- Hardware configuration settings.
- Memory map configuration settings.
- Monitor configuration settings.

Command File Command

```
FIL(E) CON(FIGURATION) STO(RE) filename
```

See Also

"To save the current emulator configuration" in the "Saving and Loading Configurations" section of the "Configuring the Emulator" chapter.

File→Copy Destination... (ALT, F, P)

Names the listing file to which debugger information may be copied.

The contents of most of the debugger windows can be copied to the destination listing file by choosing the Copy→Window command from the window's control menu.

The Symbol and Trace windows' control menus provide the Copy→All command for copying all of the symbolic or trace information to the destination listing file.

This command opens a file selection dialog box from which you select the name of the output list file. Output list files have the extension ".LST".

Command File Command

```
COP(Y) TO filename
```

See Also

"To change the list file destination" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

File→Exit (ALT, F, X)

Exits the debugger.

Command File Command

```
QUI(T)
```

See Also

"To exit the debugger" in the "Starting and Exiting the Debugger" section of the "Using the Debugger Interface" chapter.

File→Exit HW Locked (ALT, F, H)

File→Exit HW Locked (ALT, F, H)

Exits the debugger and locks the emulator hardware.

When the emulator hardware is locked, your user name and ID are saved in the HP 64700 and other users are prevented from accessing it.

You can restart the debugger and resume your debug session after reloading the symbolic information with the File→Load Object... (ALT, F, L) command.

If you have any breakpoints set when you exit the debugger, you will have to reset the breakpoints when you restart the debugger. All breakpoints are deleted when RTC is exited.

Command File Command

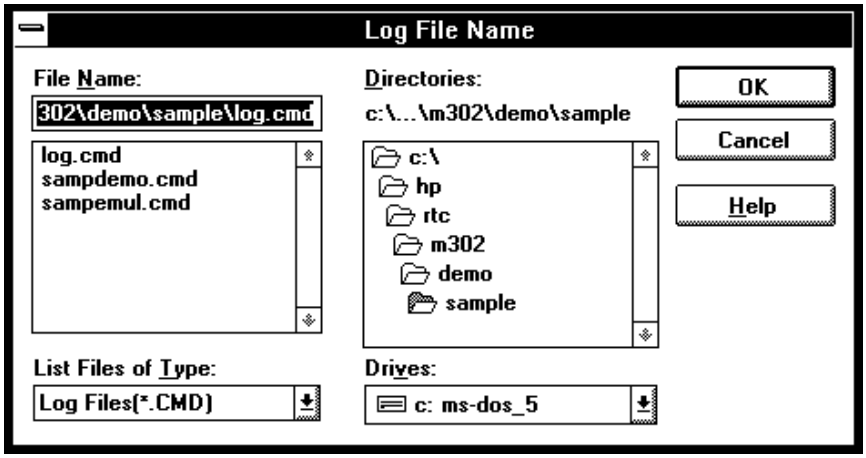
```
QUI(T) LOC(KED)
```

See Also

Settings→Communication... (ALT, S, C)

File Selection Dialog Boxes

File selection dialog boxes are used with several of the debugger commands. An example of a file selection dialog box is shown below.



File Name	You can select the name of the file from the list box and edit it in the text box.
List Files of Type	Lets you choose the filter for files shown in the File Name list box.
Directories	You can select the directory from the list box. The selected directory is shown above the list box.
Drives	Lets you select the drive name whose directories are shown in the Directories list box.
OK	Selects the named file and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.
Help	If this button is available, it opens a help window for viewing the associated help information.

Execution→Run (F5), (ALT, E, U)

Runs the program from the current program counter address.

Command File Command

RUN

Execution→Run to Cursor (ALT, E, C)

Runs from the current program counter address up to the Source window line that contains the cursor.

This command sets a breakpoint at the cursor-selected source line and runs from the current program counter address; therefore, it cannot be used when programs are in target system ROM.

If the cursor-selected source line is not reached within the number of milliseconds specified by StepTimerLen in the B3621.INI file, a dialog box appears from which you can cancel the command. When the Stop button is chosen, program execution stops, the breakpoint is deleted, and the processor continues RUNNING IN USER PROGRAM.

Command File Command

```
COM(E) address
```

See Also

"To run the program until the specified line" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Run to Caller (ALT, E, T)

Executes the user program until the current function returns to its caller.

Because this command determines the address at which to stop execution based on stack frame data and object file function information, the following restrictions are imposed:

- A function cannot properly return immediately after its entry point because the stack frame for the function has not yet been generated. Use the Step command to single-step the function before using the Execution→Run to Caller (ALT, E, T) command.
- An assembly language routine cannot properly return, even it follows C function call conventions, because there is no function information in the object file.
- An interrupt function cannot properly return because it uses a stack in a different fashion from standard functions.

Command File Command

```
RET(URN)
```

See Also

"To run the program until the current function return" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Run... (ALT, E, R)

Executes the user program starting from the specified address.

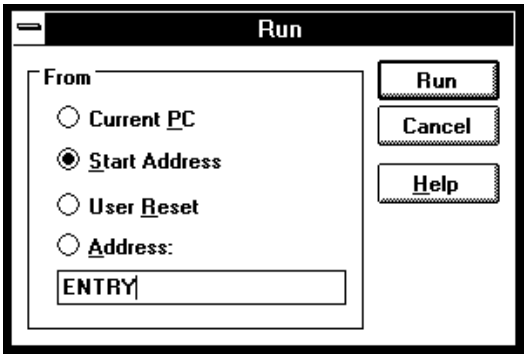
This command sets the processor status to RUNNING IN USER PROGRAM.

Note

If you try to run from an address whose symbol is START, STA, RESET, or RES (or any upper- or lower-case variation), the debugger instead runs from the start address or reset address, respectively, because these are the keywords used with the RUN command. To fix this problem, use START+0, STA+0, RESET+0, or RES+0 to force the symbol to be evaluated as an address.

Run Dialog Box

Choosing the Execution→Run... (ALT, E, R) command opens the following dialog box:



Current PC — Specifies that the program run from the current program counter address.

Start Address — Specifies that the program run from the *transfer address* defined in the object file.

User Reset	The emulator drives the target reset line and begins executing from the contents of exception vector 0 (this will occur within a few cycles of the /RESET signal).
Address	Lets you enter the address from which to run. Because the function code is determined from the memory map, do not include one with the address.
Run	Initiates program execution from the specified address, then close the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

`RUN`
Executes the user program from the current program counter address.

`RUN STA(RT)`
Executes the user program from the transfer address defined in the object file.

`RUN RES(ET)`
Drives the target reset line and begins executing from the contents of exception vector 0.

`RUN address`
Executes the user program from the specified address.

See Also

"To run the program from a specified address" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Single Step (F2), (ALT, E, N)

Executes a single instruction or source line at the current program counter address.

A single source line is executed when in the source only display mode, unless no source is available or an assembly language program is loaded; in these cases, a single assembly language instruction is executed.

When in the mnemonic mixed display mode, a single assembly language instruction is executed.

Command File Command

STE(P)

See Also

"To step a single line or instruction" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Step Over (ALT, E, O)
Execution→Step... (ALT, E, S)

Execution→Step Over (F3), (ALT, E, O)

Executes a single instruction or source line at the current program counter except when the instruction or source line makes a subroutine or function call, in which case the entire subroutine or function is executed.

This command is the same as the Execution→Single Step (ALT, E, N) command except when the source line contains a function call or the assembly instruction makes a subroutine call (with the BSR or JSR instructions). In these cases, the entire function or subroutine is executed.

Command File Command

```
OVE(R)
```

See Also

"To step over a function" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Step... (ALT, E, S)

Single-steps the specified number of instructions or source lines, starting from the specified address.

Single source lines are executed when in the source only display mode, unless no source is available or an assembly language program is loaded; in these cases, single assembly language instructions are executed.

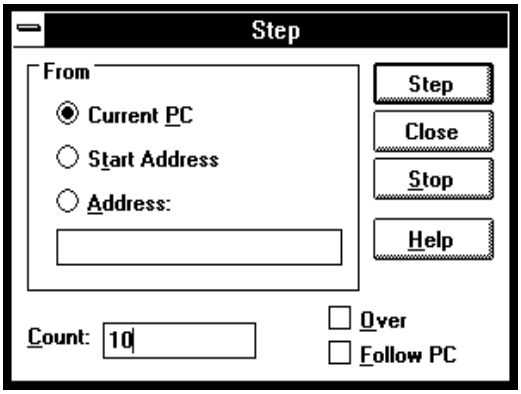
When in the mnemonic mixed display mode, single assembly language instructions are executed.

Note

If you try to step from an address whose symbol is START or STA (or any upper- or lower-case variation), the debugger instead steps from the start address because these are the keywords used with the STEP and OVER commands. To fix this problem, use START+0 or STA+0 to force the symbol to be evaluated as an address.

Step Dialog Box

Choosing the Execution→Step... (ALT, E, S) command opens the following dialog box:



Current PC	Specifies that stepping start from the current program counter address.
Start Address	Specifies that stepping start from the start address or *transfer address*.
Address	Lets you enter the address from which to single-step.
Count	Indicates the step count. The count decrements by one for every step and stops at 1.
Over	If the source line to be executed contains a function call or the assembly language instruction to be executed contains a subroutine call, this option specifies that the entire function or subroutine be executed.
Follow PC	If you check the Follow PC box, stepping will provide more detail because it will follow the PC for each step, and update the Source window after each step. Leaving this box unchecked speeds the stepping process; the steps will be counted, but the content of the Source window will not be updated until stepping is completed.

Step	Single-steps the specified number of instructions or source lines, starting from the specified address.
Close	Closes the dialog box.
Stop	Stops single-stepping.

Command File Command

`STE(P) count`
Single-steps the specified number of instructions or source lines, starting from the current program counter address.

`STE(P) count address`
Single-steps the specified number of instructions or source lines, starting from the specified address.

`STE(P) count STA(RT)`
Single-steps the specified number of instructions or source lines, starting from the transfer address defined in the object file.

`OVE(R) count`
Single-steps the specified number of instructions or source lines, starting from the current program counter address. If an instruction or source line makes a subroutine or function call, the entire subroutine or function is executed.

`OVE(R) count address`
Single-steps the specified number of instructions or source lines, starting from the specified address. If an instruction or source line makes a subroutine or function call, the entire subroutine or function is executed.

`OVE(R) count STA(RT)`
Single-steps the specified number of instructions or source lines, starting from the transfer address defined in the object file. If an instruction or source line makes a subroutine or function call, the entire subroutine or function is executed.

See Also

"To step multiple lines or instructions" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Single Step (ALT, E, N)
Execution→Step Over (ALT, E, O)

Execution→Break (F4), (ALT, E, B)

Stop user program execution and break into the monitor.

This command can also be used to break into the monitor when the processor is in the EMULATION RESET status.

Once the command has been completed, the processor transfers to the RUNNING IN MONITOR status.

Command File Command

BRE(AK)

See Also

"To stop program execution" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Reset (ALT, E, E)

Resets the emulation microprocessor.

If a foreground monitor is being used, it will automatically be loaded when this command is chosen.

While the processor is in the EMULATION RESET state, no display or modification is allowed for the contents of target system memory or registers. Therefore, before you can display or modify target system memory or processor registers, you must use the Execution→Break (ALT, E, B) command to break into the monitor.

Note that if the RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command is chosen, the emulation microprocessor may switch immediately from reset to running in monitor, for example, to update the contents of a register window.

Command File Command

`RES(ET)`

See Also

"To reset the processor" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Breakpoint→Set at Cursor (ALT, B, S)

Sets a breakpoint at the cursor-selected address in the Source window.

The breakpoint marker "BP" appears on lines at which breakpoints are set.

When a breakpoint is hit, program execution stops immediately before executing the instruction or source code line at which the breakpoint is set.

A set breakpoint remains active until it is deleted.

Because breakpoints are set by replacing program opcodes with breakpoint instructions, they cannot be set in programs stored in target system ROM. In addition, breakpoints do not function properly when set at addresses where no opcode is found.

The TRAP instruction is used as the breakpoint instruction. The TRAP number is specified with the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.

The Breakpoint→Set at Cursor (ALT, B, S) command replaces the original instruction at the specified address with a TRAP instruction. When the emulator detects the TRAP instruction, it breaks to the monitor and restores the original instruction. When the emulator detects a TRAP instruction that was not inserted as a breakpoint, the emulator breaks and transfers to the "UNDEFINED BREAKPOINT at address" status.

The Breakpoint→Set at Cursor (ALT, B, S) command may cause BP markers to appear at two or more addresses. This happens when a single instruction is associated with two or more source lines. You can select the mnemonic display mode to verify that the breakpoint is set at a single address.

Command File Command

```
BP SET address
```

See Also

"To set a breakpoint" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

Breakpoint→Delete at Cursor (ALT, B, D)

Deletes the breakpoint set at the cursor-selected address in the Source window.

This command is only applicable to lines that contain "BP" markers (which indicate set breakpoints). Once the breakpoint is deleted, the original instruction is replaced.

Command File Command

```
BP DEL(ETE) address
```

See Also

"To delete a single breakpoint" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

Breakpoint→Edit... (ALT, B, E)

Breakpoint→Set Macro... (ALT, B, M)

Sets a *break macro* immediately before the cursor-selected address in the Source window.

Break macro lines are marked with the "BP" breakpoint marker, and the corresponding addresses or line numbers are displayed in decimal format.

When a break macro is hit, program execution stops immediately before executing the instruction or source code line at which the break macro is set. Then, the commands associated with the break macro are executed. When a "RUN" command is set as the last command in the break macro, the system executes the break macro and resumes program execution.

The break macro remains active until it is deleted with the Breakpoint→Delete Macro (ALT, B, L) command or the Breakpoint→Edit... (ALT, B, E) command.

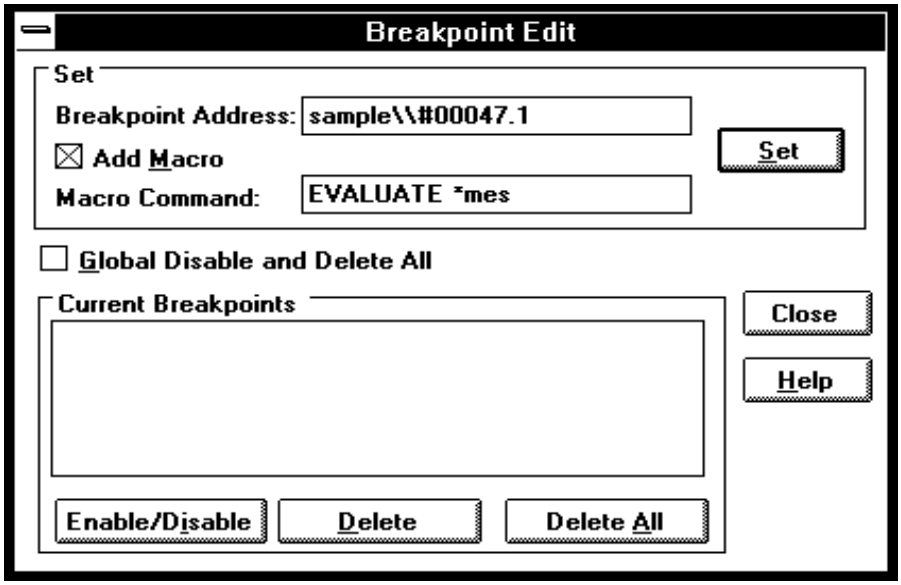
Because break macros use breakpoints, they cannot be set at addresses in target system ROM.

Additional commands can be added to existing break macros as follows:

- When a source code line or disassembled instruction is cursor-selected, the additional command is inserted at the top of the list of commands.
- When a macro command line is cursor-selected, the additional command is inserted immediately following the cursor-selected command.

Breakpoint Edit Dialog Box

Choosing the Breakpoint→Set Macro... (ALT, B, M) command opens the following dialog box:



Breakpoint Address	Displays the specified line number or address followed by a decimal point and the break macro line number.
Add Macro	Activates the Macro Command text box.
Macro Command	Specifies the command to be added to the break macro.
Set	Inserts the specified macro command at the location immediately preceding the specified source line or address, or inserts the macro command at the location immediately following the specified break macro line. Two or more commands can be associated with a break macro by entering the first command and choosing Set, then entering the second command and choosing Set, and so on. Commands execute in the order of their entry.

Global Disable and Delete All	Disables and deletes all current breakpoints and break macros.
Current Breakpoints	Displays the addresses and line numbers of the current breakpoints and break macros. Allows you to select breakpoints or break macros to be deleted.
Enable/Disable	Enable/Disable the selected breakpoint and break macro.
Delete	Deletes the selected breakpoints or break macros from the Current Breakpoints list box.
Delete All	Deletes all breakpoints and break macros from the Current Breakpoints list box.
Close	Closes the dialog box.

Command File Command

```
BM SET address command
```

See Also

"To set a break macro" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

Breakpoint→Delete Macro (ALT, B, L)

Removes the break macro set at the cursor-indicated address in the Source window.

This command is only applicable to lines that contain "BP" markers (which indicate set breakpoints) or break macro lines.

When a source code line is cursor-selected, this command removes the breakpoint and all the macros commands set at the line.

When a break macro line is cursor-selected, this command removes the single macro command at the line.

Command File Command

```
BM DEL(ETE) address
```

See Also

"To delete a single break macro" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

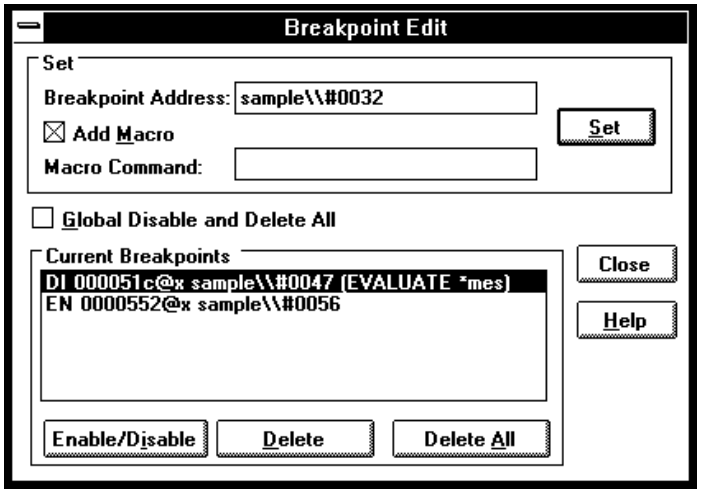
Breakpoint→Edit... (ALT, B, E)

Breakpoint→Edit... (ALT, B, E)

Lets you set, list, or delete breakpoints and break macros. Breakpoints are always globally enabled on initial entry into the RTC interface.

Breakpoint Edit Dialog Box

Choosing the Breakpoint→Edit... (ALT, B, E) command opens the following dialog box:



Breakpoint Address	Lets you specify the address at which to set a breakpoint or a break macro.
Add Macro	When selected, this specifies that a break macro should be included with the breakpoint.
Macro Command	Lets you specify the macro to be included with the breakpoint.
Set	Sets a breakpoint with or without a break macro at the specified address.

Global Disable and Delete All	When selected, all existing breakpoints are deleted (not simply disabled), and no new breakpoints can be added.
Current Breakpoints	Displays the addresses and line numbers of the current breakpoints and break macros. Allows you to select the breakpoints or break macros to be enabled/disabled or deleted.
Enable/Disable	Disables or enables the selected breakpoints or breakpoint macros in the Current Breakpoints list box. Enabled breakpoints begin with EN in the Current Breakpoints list and show "BP" at the start of the line in the Source window list. Disabled breakpoints begin with DI in the Current Breakpoints list and show "bp" at the start of the line in the Source window list.
Delete	Deletes the selected breakpoints or break macros from the Current Breakpoints list box.
Delete All	Deletes all the breakpoints and break macros from the Current Breakpoints list box.
Close	Closes the dialog box.

Command File Command

```
MOD(E) BKP(TBREAK) ON|OFF

BP DEL(ETE) ALL

BP DIS(ABLE) address

BP ENA(BLE) address
```

See Also

"To disable a breakpoint" and
"To list the breakpoints and break macros" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

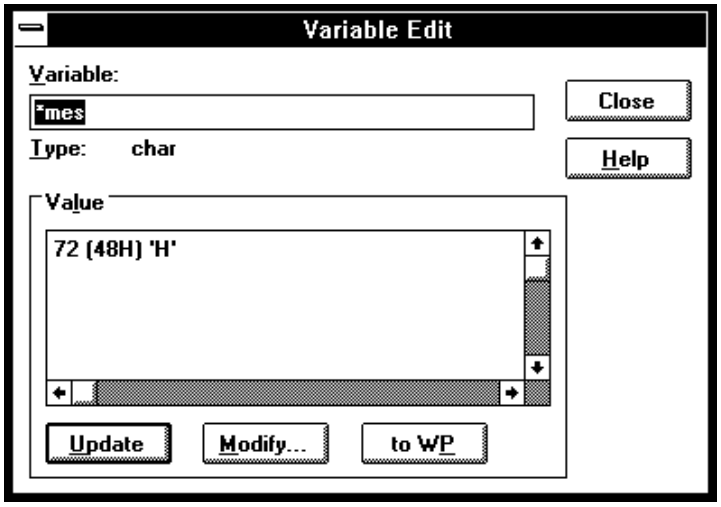
Variable→Edit... (ALT, V, E)

Displays or modifies the contents of the specified variable or copies it to the WatchPoint window.

A dynamic variable can be registered as a watchpoint when the current program counter is in the function in which the variable is declared. If the program counter is not in this function, the variable name is invalid and an error results.

Variable Edit Dialog Box

Choosing the Variable→Edit... (ALT, V, E) command opens the following dialog box:



Variable	Specifies the name of the variable to be displayed or modified. The contents of the clipboard, usually a variable selected from the another window, automatically appears in this text box.
Type	Displays the type of the specified variable.
Value	Displays the contents of the specified variable.

Update	Reads and displays the contents of the variable specified in the Variable text box.
Modify	Modifies the contents of the specified variable. Choosing this button opens the Variable Modify Dialog Box, which lets you edit the contents of the variable.
to WP	Adds the specified variable to the WatchPoint window.
Close	Closes the dialog box.

Command File Command

`VARI(ABLE) variable TO data`
Replaces the contents of the specified variable with the specified value.

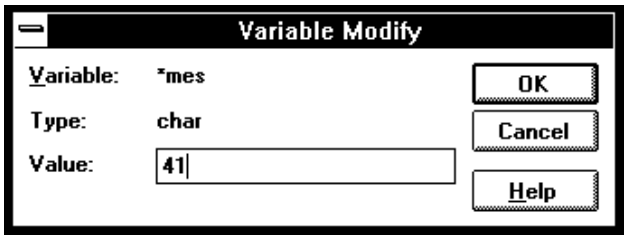
See Also

"To display a variable" and
"To monitor a variable in the WatchPoint window" in the "Displaying and Editing Variables" section of the "Debugging Programs" chapter.

"Symbols" in the "Expressions in Commands" chapter.

Variable Modify Dialog Box

Choosing the Modify button in the Variable Edit dialog box opens the following dialog box, where you enter the new value and choose the OK button to confirm the new value.



Variable	Shows the variable to be edited.
Type	Indicates the type of the variable displayed in the Variable field.
Value	Lets you enter the new value of the variable.
OK	Replaces the contents of the specified variable with the specified value and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

See Also

"To edit a variable" in the "Displaying and Editing Variables" section of the "Debugging Programs" chapter.

Trace→Function Flow (ALT, T, F)

Traces function flow by storing function entry points in the trace buffer.

The analyzer identifies function entry points by looking for the following sequence:

1. A program fetch of the LINK instruction.
2. A data write (the first word of the stack pointer push). If a non-program fetch state is captured before the data write, the sequence restarts.
3. A second data write (the second word of the stack pointer push). If any other state is captured, the sequence restarts.
4. Any program fetch. The sequence is repeated to identify the next function entry point.

Assembly language functions can also be traced provided that they comply with C function call conventions.

Note

When using the MCC68K compiler, you must specify the -Kf option when compiling programs in order for the debugger to be able to trace function flow. (The -Kf option creates frame pointers for functions.)

Command File Command

```
TRA(CE) FUN(CTION) FLO(W)
```

See Also

"To trace function flow" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Function Caller... (ALT, T, C)

Traces the caller of the specified function.

The function name can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

The analyzer stores only the execution of the function entry point and prestores execution states that occur before the function entry point. These prestored states correspond to the function call statements and identify the caller of the function.

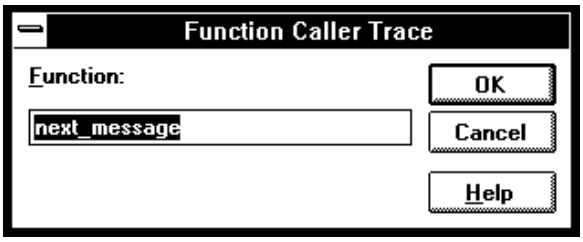
When assembly language programs are used, you can specify the assembler symbol for a subroutine instead of a C function name, and the prestored states will show the instructions that called the subroutine.

Note

Because of prefetching by the 68302 processor, the analyzer may fail in tracing the caller.

Function Caller Trace Dialog Box

Choosing the Trace→Function Caller... (ALT, T, C) command opens the following dialog box:



Function	Lets you enter the function whose callers you want to trace.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) FUNC(TION) CAL(L) address
```

See Also

"To trace callers of a specified function" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Function Statement... (ALT, T, S)

Traces execution within the specified function.

The function name can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

The analyzer stores execution states in the function's address range.

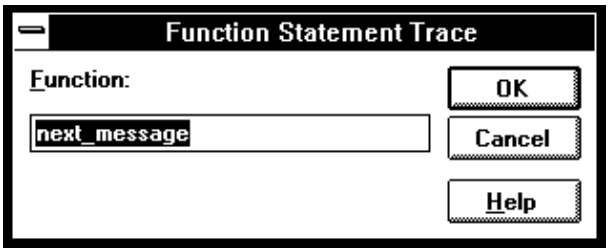
Because the analyzer is set up based on function information from the object file, this command cannot be used to trace non-C functions.

Note

The analyzer traces unexecuted instructions due to prefetching by 68302 processor.

Function Statement Trace Dialog Box

Choosing the Trace→Function Statement... (ALT, T, S) command opens the following dialog box:



Function	Lets you enter the function whose execution you want to trace.
OK	Traces within the specified function and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) FUNC(TION) STA(TEMENT) address
```

See Also

"To trace execution within a specified function" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Variable Access... (ALT, T, V)

Traces accesses to the specified variable.

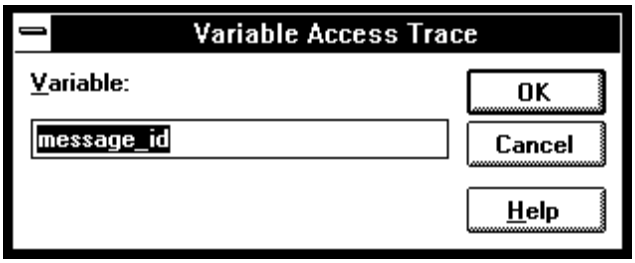
The variable name can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

You can specify any of the external or static variables, or the variables having a fixed address throughout the course of program execution.

The analyzer stores only accesses within the range of the variable and prestores execution states that occur before the access. These prestored states correspond to the statements that access the variable.

Variable Access Dialog Box

Choosing the Trace→Variable Access... (ALT, T, V) command opens the following dialog box:



Variable	Lets you enter the variable name.
OK	Traces accesses to the specified variable and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) VAR(IABLE) ACC(ESS) address
```

See Also

"To trace accesses to a specified variable" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Variable Break... (ALT, T, B)

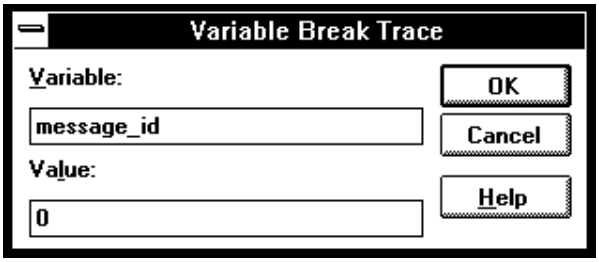
Traces before, and breaks program execution when, a value is written to a variable.

The variable name can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

You can specify any of the external or static variables, or the variables having a fixed address throughout the course of program execution.

Variable Break Dialog Box

Choosing the Trace→Variable Break... (ALT, T, B) command opens the following dialog box:



Variable	Lets you enter the variable name.
Value	Lets you enter the value that, when written to the variable, triggers the analyzer.
OK	Starts the trace and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) VAR(IABLE) BRE(AK) address data
```

See Also

"To trace before a particular variable value and break" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Edit... (ALT, T, E)

Edits the trace specification of the last trace command.

This command is useful for making modifications to the last entered trace command, even if the analyzer was setup automatically as with the Trace→Function or Trace→Variable commands.

Trace specifications are edited with Sequence Trace Setting dialog box.

Command File Command

`TRA(CE) SAV(E) filename`
Stores the current trace specification to a file.

`TRA(CE) LOA(D) filename`
Loads the specified trace setting file.

`TRA(CE) CUS(TOMIZE)`
Traces program execution using the loaded trace setting file.

See Also

"To edit a trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Sequence... (ALT, T, Q)

Trace→Trigger Store... (ALT, T, T)

Traces program execution as specified in the Trigger Store Trace dialog box.

You can enter address, data, and status values that qualify the state(s) that, when captured by the analyzer, will be stored in the trace buffer or will trigger the analyzer.

Data values are 16-bit values (because the data bus is 16 bits wide). To identify byte values on the data bus, use "don't cares" as shown below:

Data at an even address: 12xx

Data at an odd address: 0xx34

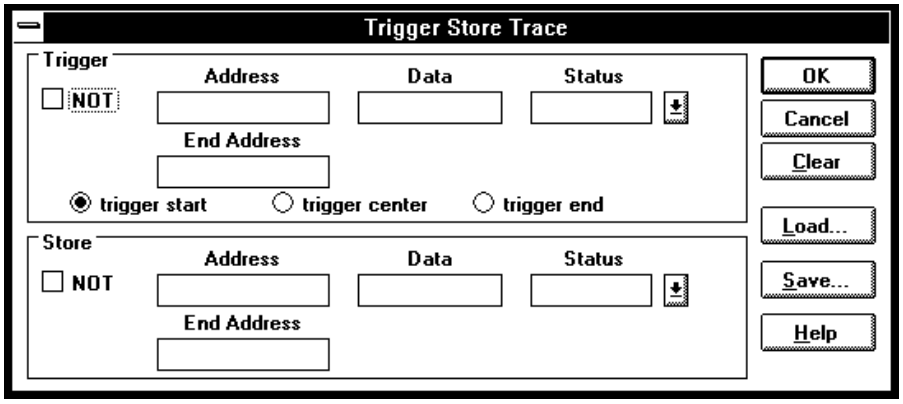
Status values identify the types of microprocessor bus cycles. You may select status values from a predefined list.

Note

The analyzer traces unexecuted instructions due to prefetching by the 68302 processor.

Trigger Store Trace Dialog Box

Choosing the Trace→Trigger Store... (ALT, T, T) opens the following dialog box:



Trigger	This box groups the items that make up the trigger condition.
NOT	Specifies any state that does not match the Address, Data, and Status values.
Address	Specifies the address portion of the state qualifier.
End Address	Specifies the end address of an address range.
Data	Specifies the data portion of the state qualifier.
Status	Specifies the status portion of the state qualifier.
trigger start	Specifies that states captured after the trigger condition be stored in the trace buffer.
trigger center	Specifies that states captured before and after the trigger condition be stored in the trace buffer.
trigger end	Specifies that states captured before the trigger condition be stored in the trace buffer.
Store	This box groups the items that make up the store condition.
OK	Starts the specified trace and closes the dialog box.
Cancel	Cancels the trace setting and closes the dialog box.
Clear	Restores the dialog box to its default state.
Load...	Opens a file selection dialog box from which you select the name of a trace specification file previously saved from the Trigger Store Trace dialog box. Trace specification files have the extension ".TRC".
Save...	Opens a file selection dialog box from which you select the name of the trace specification file.

Command File Command

`TRA(CE) LOA(D) filename`
Loads the specified trace setting file.

`TRA(CE) CUS(TOMIZE)`
Traces program execution using the loaded trace setting file.

See Also

"To set up a 'Trigger Store' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)

Traces program execution as specified in the Find Then Trigger Trace dialog box.

This command lets you set up a two level sequential trace specification that works like this:

1. Once the trace starts, the analyzer stores (in the trace buffer) the states that satisfy the Enable Store condition while searching for a state that satisfies the Enable condition.
2. After the Enable condition has been found, the analyzer stores the states that satisfy the Trigger Store condition while searching for a state that satisfies the Trigger condition.
3. After the Trigger condition has been found, the analyzer stores the states that satisfy the Store condition.

If any state during the sequence satisfies the Restart condition, the sequence starts over.

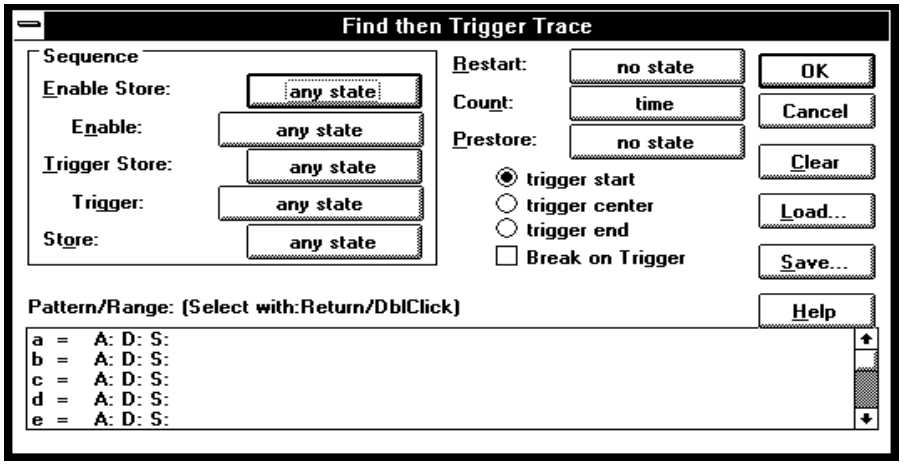
You can enter address, data, and status values that qualify state(s) by setting up pattern or range resources. These patterns and range resources are used when defining the various conditions.

Note

The analyzer traces unexecuted instructions due to prefetching by the 68302 processor.

Find Then Trigger Trace Dialog Box

Choosing the Trace→Find Then Trigger... (ALT, T, D) command opens the following dialog box:



The Sequence group box specifies a two term sequential trigger condition. It also lets you specify store conditions during the sequence.

Enable Store	Qualifies the states that get stored (in the trace buffer) while searching for a state that satisfies the enable condition.
Enable	Specifies the condition that causes a transfer to the next sequence level.
Trigger Store	Qualifies the states that get stored while the analyzer searches for the trigger condition.
Trigger	Specifies the trigger condition.
Store	Qualifies the states that get stored after the trigger condition is found.
Restart	Specifies the condition that restarts the sequence.

Count	Specifies whether time or the occurrences of a particular state are counted; you can also turn counts OFF. See the Condition Dialog Boxes.
Prestore	Qualifies the states that may be stored before each normally stored state. Up to two states may be prestored for each normally stored state. Prestored states can be used to show from where a function is called or a variable is accessed.
trigger start	The state that satisfies trigger condition is positioned at the start of the trace, and states that satisfy the Store condition will be stored after the trigger. In this case, the states that satisfy the Enable Store and Trigger Store conditions will not appear in the trace.
trigger center	The state that satisfies the trigger condition is positioned in the center of the trace, and states that satisfy the store conditions will be stored before and after the trigger.
trigger end	The state that satisfies the trigger condition is positioned at the end of the trace, and states that satisfy the Enable Store and Trigger Store conditions will be stored before the trigger. In this case, states that satisfy the Store condition will not appear in the trace.
Break on Trigger	When selected, this option specifies that execution break into the monitor when the analyzer is triggered.
Pattern/Range	Specifies the trace patterns for the state conditions. Double-clicking the desired pattern in the Pattern/Range list box opens the Trace Pattern Dialog Box or the Trace Range Dialog Box, where you specify the desired trace pattern or range. Clicking the Sequence, Restart, Count, or Prestore buttons causes the Condition Dialog Boxes to be opened. This dialog box lets you select or combine patterns or ranges to specify the condition.

OK	Starts the specified trace and closes the dialog box.
Cancel	Cancels trace setting and closes the dialog box.
Clear	Restores the dialog box to its default state.
Load...	Opens a file selection dialog box from which you select the name of a trace specification file previously saved from the Trigger Store Trace or Find Then Trigger Trace dialog boxes. Trace specification files have the extension ".TRC".
Save...	Opens a file selection dialog box from which you select the name of the trace specification file.

Command File Command

`TRA(CE) LOA(D) filename`
Loads the specified trace setting file.

`TRA(CE) CUS(TOMIZE)`
Traces program execution using the loaded trace setting file.

See Also

"To set up a 'Find Then Trigger' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Sequence... (ALT, T, Q)

Traces program execution as specified in the Sequence Trace dialog box.

This command lets you set up a multilevel sequential trace specification that works like this:

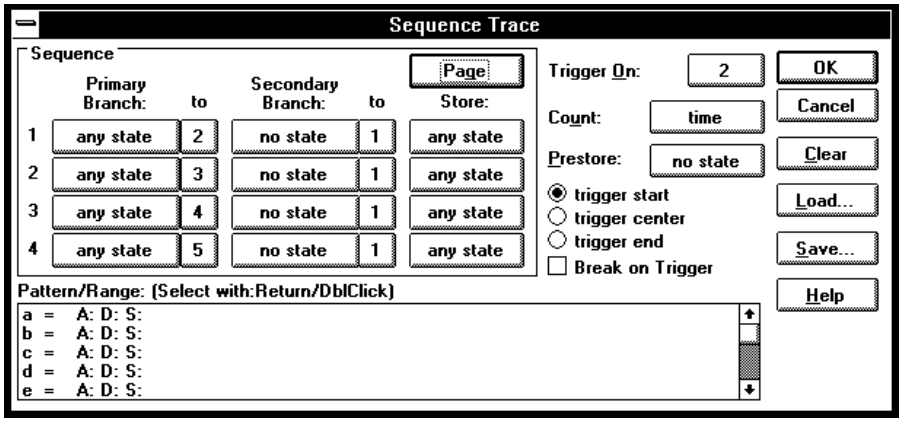
1. Once the trace starts, the analyzer stays on sequence level 1 until the primary or secondary branch condition is found. (If a state satisfies both primary and secondary branch conditions, the primary branch is taken.) Once the primary or secondary branch condition is found, the analyzer transfers to the sequence level specified by the "to" button.
2. The analyzer stays at the next sequence level until its primary or secondary branch condition is met; then, the analyzer transfers to the sequence level specified by the "to" button.
3. When the analyzer reaches the sequence level specified in Trigger On, the analyzer is triggered.
4. During the above described operation, the analyzer stores the states specified in the Store text box.

Note

The analyzer traces unexecuted instructions due to prefetching by the 68302 processor.

Sequence Trace Dialog Box

Choosing the Trace→Sequence... (ALT, T, Q) command opens the following dialog box:



The Sequence group box specifies two types of branch conditions for transferring from one sequence level to another. It also specifies store conditions for each of sequence levels 1 through 8.

Primary Branch	Specifies the condition for transferring to the sequence level specified in the "to" text box.
Secondary Branch	Specifies the condition for transferring to the sequence level specified in the "to" text box. Secondary branches are used to do things like restart the sequence if a particular state is found.
Store	Specifies the states stored in the trace buffer at each sequence level.
Page	Toggles the display between sequence levels 1 through 4 and levels 5 through 8.
Trigger On	Specifies the sequence level whose entry triggers the analyzer. See the Sequence Number Dialog Box.

Count	Specifies whether time or the occurrences of a particular state are counted; you can also turn counts OFF. See the Condition Dialog Boxes.
Prestore	Qualifies the states that may be stored before each normally stored state. Up to two states may be prestored for each normally stored state. Prestored states can be used to show from where a function is called or a variable is accessed.
trigger start	The state that satisfies trigger condition is positioned at the start of the trace, and states that satisfy the store conditions will be stored after the trigger.
trigger center	The state that satisfies the trigger condition is positioned in the center of the trace, and states that satisfy the store conditions will be stored before and after the trigger.
trigger end	The state that satisfies the trigger condition is positioned at the end of the trace, and states that satisfy the store conditions will be stored before the trigger.
Break on Trigger	When selected, this option specifies that execution break into the monitor when the analyzer is triggered.
Pattern/Range	Specifies the trace patterns for the state conditions. Double-clicking the desired pattern in the Pattern/Range list box opens the Trace Pattern Dialog Box or the Trace Range Dialog Box, where you specify the desired trace pattern or range. Clicking the Primary Branch, Secondary Branch, Store, Count, or Prestore buttons causes the Condition Dialog Boxes to be opened. This dialog box lets you select or combine patterns or ranges to specify the condition.
OK	Starts the specified trace and closes the dialog box.
Cancel	Cancels trace setting and closes the dialog box.

Clear	Restores the dialog box to its default state.
Load...	Opens a file selection dialog box from which you select the name of a trace specification file previously saved from any of the trace setting dialog boxes. Trace specification files have the extension ".TRC".
Save...	Opens a file selection dialog box from which you select the name of the trace specification file.

Command File Command

`TRA(CE) LOA(D) filename`
Loads the specified trace setting file.

`TRA(CE) CUS(TOMIZE)`
Traces program execution using the loaded trace setting file.

See Also

"To set up a 'Sequence' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Until Halt (ALT, T, U)

Traces program execution until the Trace→Halt (ALT, T, H) command is chosen.

This command is useful in tracing execution that leads to a processor halt or a break to the background monitor. Before executing the program, choose the Trace→Until Halt (ALT, T, U) command. Then, run the program. After the processor has halted or broken into the background monitor, choose the Trace→Halt (ALT, T, H) command to stop the trace. The execution that led up to the break or halt will be displayed.

Command File Command

```
TRA(CE) ALW(AYS)
```

See Also

"To trace until the command is halted" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Halt (ALT, T, H)

Stops a running trace.

This command stops a currently running trace whether the trace was started with the Trace→Until Halt (ALT, T, U) command or another trace command.

As soon as the analyzer stops the trace, stored states are displayed in the Trace window.

Command File Command

```
TRA(CE) STO(P)
```

See Also

"To stop a running trace" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Again (F7), (ALT, T, A)

Traces program execution using the last trace specification stored in the HP 64700.

If you haven't entered a trace command since you started the debugger, the last trace specification stored in the HP 64700 may be a trace specification set up by a different user; in this case, you cannot view or edit the trace specification.

Command File Command

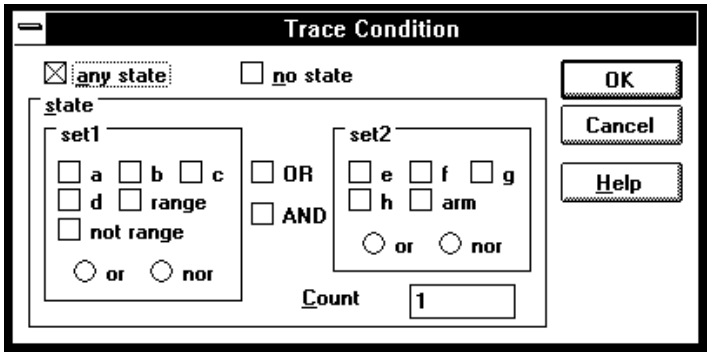
```
TRA(CE) AGA(IN)
```

See Also

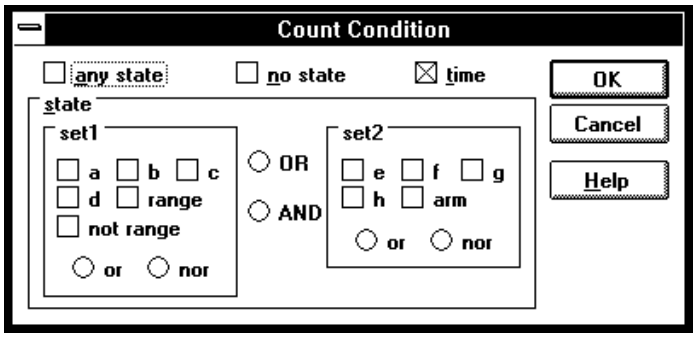
"To repeat the last trace" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Condition Dialog Boxes

Choosing the buttons associated with enable, trigger, primary branch, secondary branch, store, or prestore conditions opens the following dialog box:



Choosing the button associated with the count condition opens the following dialog box:



no state	No state meets the specified condition.
any state	Any state meets the specified condition.
time	The analyzer counts time for each state stored in the trace.

state

This group box lets you qualify the state that will meet the specified condition. You can qualify the state as one of the patterns "a" through "h", the "range", or the "arm", or you can qualify the state as a combination of the patterns, range, or arm by using the interset or intraset operators.

a b c d e f g h

The patterns that qualify states by identifying the address, data, and/or status values.

The values for a pattern are specified by selecting one of the patterns in the Pattern/Range list box and entering values in the Trace Pattern Dialog Box.

range

Identifies a range of address or data values.

The values for a range are specified by selecting the range in the Pattern/Range list box and entering values in the Trace Range Dialog Box.

not range

Identifies all values not in the specified range.

arm

Identifies the condition that arms (in other words, activates) the analyzer. The analyzer can be armed by an input signal on the BNC port.

or/nor

You can combine patterns within the set1 or set2 group boxes with these logical operators.

You can create the AND and NAND operators by selecting NOT when defining patterns and applying DeMorgan's law (the / character is used to represent a logical NOT):

```
AND      A and B  =  /(/A or /B)   NOR
NAND   /(A and B) =    /A or /B    OR
```

OR/AND

You can combine patterns from the set1 and set2 group boxes with these logical operators.

Count	Appearing in Trace Condition dialog boxes, this value specifies the number of occurrences of the state that will satisfy the condition.
OK	Applies the state qualifier to the specified condition and closes the dialog box.
Cancel	Closes the dialog box.

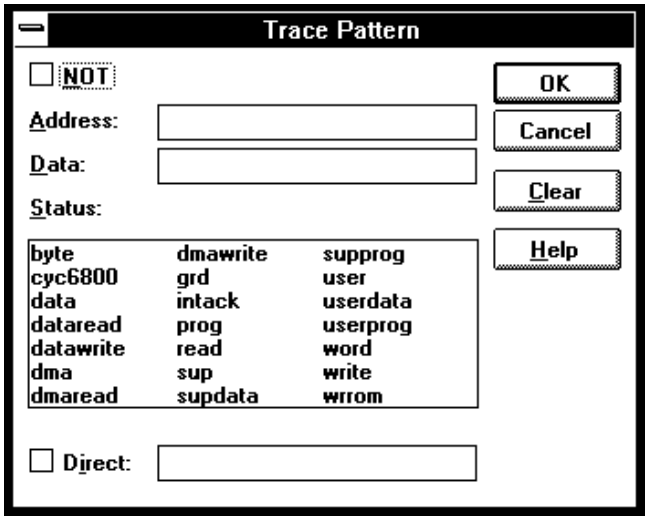
See Also

"To set up a 'Find Then Trigger' trace specification" and
"To set up a 'Sequence' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)
Trace→Sequence... (ALT, T, Q)

Trace Pattern Dialog Box

Selecting one of the patterns in the Pattern/Range list box opens the following dialog box:



NOT	Lets you specify all values other than the address, data, and/or status values specified.
Address	Lets you enter the address value for the pattern.
Data	Lets you enter the data value for the pattern.
Status	Lets you select the *status value* for the pattern.
Direct	Lets you enter a status value other than one of the predefined status values.
Clear	Clears the values specified for the pattern.
OK	Applies the values specified for the pattern, and closes the dialog box.

Cancel Closes the dialog box.

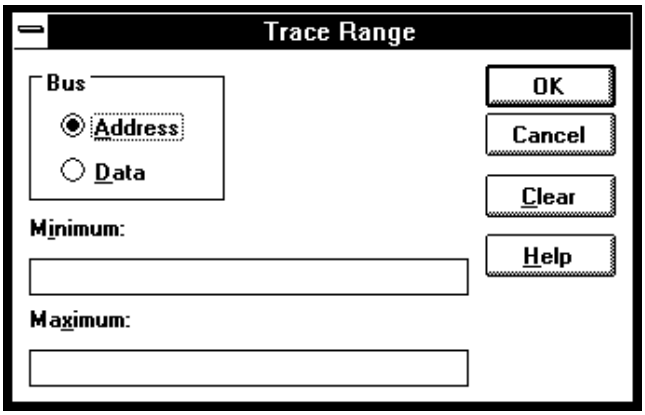
See Also

"To set up a 'Find Then Trigger' trace specification" and
"To set up a 'Sequence' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)
Trace→Sequence... (ALT, T, Q)

Trace Range Dialog Box

Selecting the range in the Pattern/Range list box opens the following dialog box:



Address	Selects a range of address values.
Data	Selects a range of data values.
Minimum	Lets you enter the minimum value for the range.
Maximum	Lets you enter the maximum value for the range.
OK	Applies the values specified for the range, and closes the dialog box.
Cancel	Closes the dialog box.
Clear	Clears the values specified for the range.

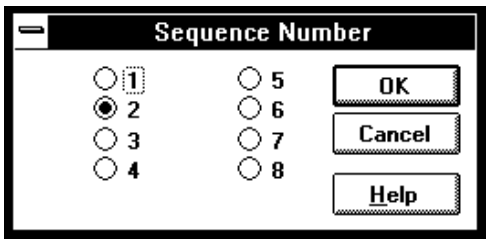
See Also

"To set up a 'Find Then Trigger' trace specification" and
"To set up a 'Sequence' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)
Trace→Sequence... (ALT, T, Q)

Sequence Number Dialog Box

Choosing the buttons associated with "to" or Trigger On opens the following dialog box:



1-8	These options specify the sequence level.
OK	Applies the selected sequence level and closes the dialog box.
Cancel	Closes the dialog box.

See Also

"To set up a 'Sequence' trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Sequence... (ALT, T, Q)

RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D)

Activates the real-time mode.

When the user program is running in real-time mode, no command that would normally cause temporary suspension of program execution is allowed. Also, the system hides:

- The Register window.
- Target system memory in the Memory window.
- Target system I/O locations in the I/O window.
- Target system memory variables in the WatchPoint window.
- Target system memory in the Source window.

While the processor is in the RUNNING REALTIME IN USER PROGRAM state, no display or modification is allowed for the contents of target system memory or registers. Therefore, before you can display or modify target system memory or processor registers, you must use the Execution→Break (ALT, E, B) command to stop user program execution and break into the monitor.

Command File Command

```
MOD(E) REA(LTIME) ON
```

See Also

"To allow or deny monitor intrusion" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Monitor Intrusion→Allowed (ALT, R, T, A)

Deactivates the real-time mode.

Commands that cause temporary breaks to the monitor during program execution are allowed.

Command File Command

```
MOD(E) REA(LTIME) OFF
```

See Also

"To allow or deny monitor intrusion" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→I/O Polling→ON (ALT, R, I, O)

Enables access to I/O.

Command File Command

```
MOD(E) IOG(UARD) OFF
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→I/O Polling→OFF (ALT, R, I, F)

Disables access to I/O.

When polling is turned OFF, values in the I/O window are updated on entry to the monitor. When monitor intrusion is not allowed during program execution, the I/O window is not updated and contents are replaced by dashes (-).

Command File Command

```
MOD(E) IOG(UARD) ON
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Watchpoint Polling→ON (ALT, R, W, O)

Turns ON polling to update values displayed in the WatchPoint window.

When polling is turned ON, temporary breaks in program execution occur when the WatchPoint window is updated.

Command File Command

```
MOD(E) WAT(CHPOLL) ON
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Watchpoint Polling→OFF (ALT, R, W, F)

Turns OFF polling to update values displayed in the WatchPoint window.

When polling is turned OFF, values in the WatchPoint window are updated on entry to the monitor. When monitor intrusion is not allowed during program execution, the WatchPoint window is not updated and contents are replaced by dashes (-).

Command File Command

```
MOD(E) WAT(CHPOLL) OFF
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Memory Polling→ON (ALT, R, M, O)

Turns ON polling to update target memory values displayed in the Memory window.

When polling is turned ON, temporary breaks in program execution occur when target system memory locations in the Memory window are updated. When monitor intrusion is not allowed during program execution, the contents of target memory locations are replaced by dashes (-).

Also, when polling is turned ON, you can modify the addresses displayed or contents of memory locations by double-clicking on the address or value, using the keyboard to type in the new address or value, and pressing the Enter key.

Command File Command

```
MOD(E) MEM(ORYPOLL) ON
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Memory Polling→OFF (ALT, R, M, F)

Turns OFF polling to update target memory values displayed in the Memory window.

When polling is turned OFF, values in the Memory window are updated on entry to the monitor.

Also, when polling is turned OFF, you cannot modify the addresses displayed or contents of memory locations by double-clicking on the address or value.

Command File Command

```
MOD(E) MEM(ORYPOLL) OFF
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

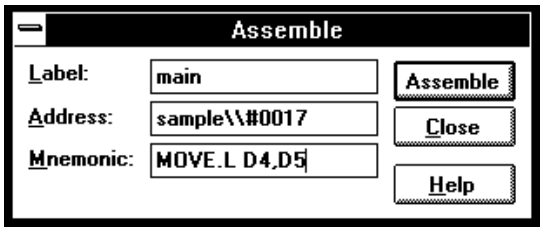
Assemble... (ALT, A)

In-line assembler.

This command lets you modify programs by specifying assembly language instructions which are assembled and loaded into program memory.

Assembler Dialog Box

Choosing the Assemble... (ALT, A) command opens the following dialog box:



Label	Lets you assign a user-defined symbol to the specified address.
Address	Lets you enter the address at which the assembly language instruction will be loaded.
Mnemonic	Lets you enter the assembly language instruction to be assembled.
Assemble	Assembles the instruction in the Mnemonic text box, and loads it into memory at the specified address.
Close	Closes the dialog box.

Command File Command

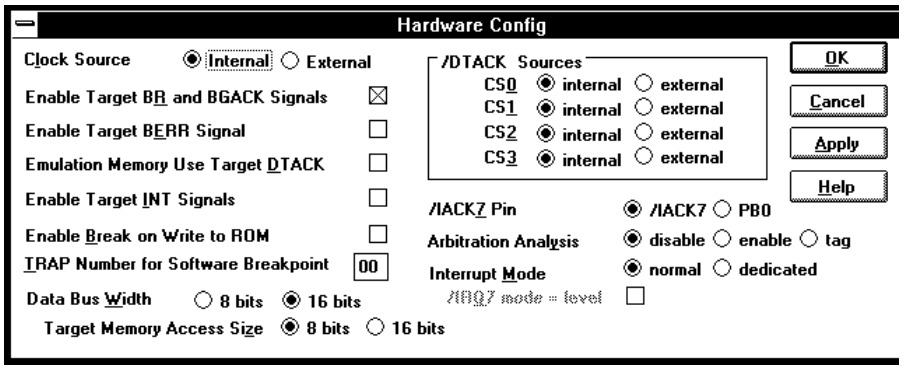
```
ASM address label "inst_string"
```

Settings→Emulator Config→Hardware... (ALT, S, E, H)

Specifies the emulator configuration.

Hardware Config Dialog Box

Choosing the Settings→Emulator Config→Hardware... (ALT, S, E, H) command opens the following dialog box:



Clock Source	Specifies the Internal or an External clock as the emulation microprocessor clock source.
Enable Target BR and BGACK Signals	Enables or disables /BR and /BGACK input from the target system.
Enable Target BERR Signal	Enables or disables /BERR input from the target system.
Emulation Memory Use Target DTACK	Enables or disables /DTACK input from the target system.
Enable Target INT Signals	Enables or disables interrupts from the target system.

Enable Break on Write to ROM	Enables or disables breaks to the monitor when the user program writes to memory mapped as ROM.
TRAP Number for Software Breakpoint	Specifies the breakpoint trap instruction number.
Data Bus Width	Specifies the size of the processor data bus.
Target Memory Access Size	Specifies the size used to access target system memory.
/DTACK Sources	Specifies, for each of the chip selects, whether the /DTACK cycle termination signal is generated internally or externally.
/IACK7 Pin	Selects the operation of this pin as either /IACK7 or PB0.
Arbitration Analysis	Specifies whether tracing of internal or external DMA bus cycles is enabled or disabled, or whether a single analyzer state be generated each time an external bus arbitration sequence occurs.
Interrupt Mode	Selects either the normal or dedicated mode for the emulation microprocessor.
/IRQ7 mode = level	When the dedicated interrupt mode is selected, this option selects whether the /IRQ7 pin is level or edge sensitive.
OK	Stores the current modification and closes the dialog box.
Cancel	Cancels the current modification and closes the dialog box.
Apply	Loads the configuration settings into the emulator.

Command File Command

`CON(FIG) CLO(CK) INT(ERNAL)`
Selects the internal clock.

`CON(FIG) CLO(CK) EXT(ERNAL)`
Selects the external clock.

`CON(FIG) BR ENA(BLE)`
Enables /BR input from the target system.

`CON(FIG) BR DIS(ABLE)`
Disables /BR input from the target system.

`CON(FIG) BER(R) EN(ABLE)`
Enables /BERR input from the target system.

`CON(FIG) BER(R) DIS(ABLE)`
Disables /BERR input from the target system.

`CON(FIG) DAC(K) EN(ABLE)`
Enables /DTACK input from the target system.

`CON(FIG) DAC(K) DIS(ABLE)`
Disables /DTACK input from the target system.

`CON(FIG) INT ENA(BLE)`
Enables interrupts from the target system.

`CON(FIG) INT DIS(ABLE)`
Disables interrupts from the target system.

`CON(FIG) ROM(BREAK) ENA(BLE)`
Enables breaks to the monitor when writes to ROM occur.

`CON(FIG) ROM(BREAK) DIS(ABLE)`
Disables breaks to the monitor when writes to ROM occur.

`CON(FIG) TRAP number`
Specifies the breakpoint TRAP instruction number.

`CON(FIG) BUS 8`
Specifies an 8-bit wide data bus.

`CON(FIG) BUS 16`
Specifies a 16-bit wide data bus.

`CON(FIG) ACC(ESS) BYT(ES)`
Specifies that target memory is accessed in byte sized locations.

`CON(FIG) ACC(ESS) WOR(DS)`
Specifies that target memory is accessed in 16-bit word sized locations.

`CON(FIG) ARB(ITRATION) ENA(BLE)`
Enables tracing of the DMA cycles.

`CON(FIG) ARB(ITRATION) DIS(ABLE)`
Disables tracing of the DMA cycles.

`CON(FIG) ARB(ITRATION) TAG`
Enables a single state in the trace each time an external bus arbitration sequence occurs.

`CON(FIG) INTMODE NOR(MAL)`
Selectes the normal interrupt mode for the emulation microprocessor.

`CON(FIG) INTMODE DED(ICATED)`
Selectes the dedicated interrupt mode for the emulation microprocessor.

`CON(FIG) IAC(K7) PB0`
Selects the PB0 pin.

`CON(FIG) IAC(K7) IAC(K7)`
Selects the IACK7 pin.

`CON(FIG) CS0/CS1/CS2/CS3 EXT(ERNAL)`
Specifies that the /DTACK cycle termination signal is generated externally.

`CON(FIG) CS0/CS1/CS2/CS3 INT(ERNAL)`
Specifies that the /DTACK cycle termination signal is generated internally.

Any of the above command file commands must be preceded and followed by the respective start and end commands:

`CON(FIG) STA(RT)`
Starts the configuration option command section.

`CON(FIG) END`
Ends the configuration option command section.

See Also

"Setting the Hardware Options" in the "Configuring the Emulator" chapter.

Settings→Emulator Config→Memory Map... (ALT, S, E, M)

Maps memory ranges.

You can map up to 7 address ranges (map terms). The minimum amount of emulation memory that can be allocated to a range is 512 bytes for 128 Kbyte memory boards and 1024 bytes for 512 Kbyte memory boards.

You can map ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

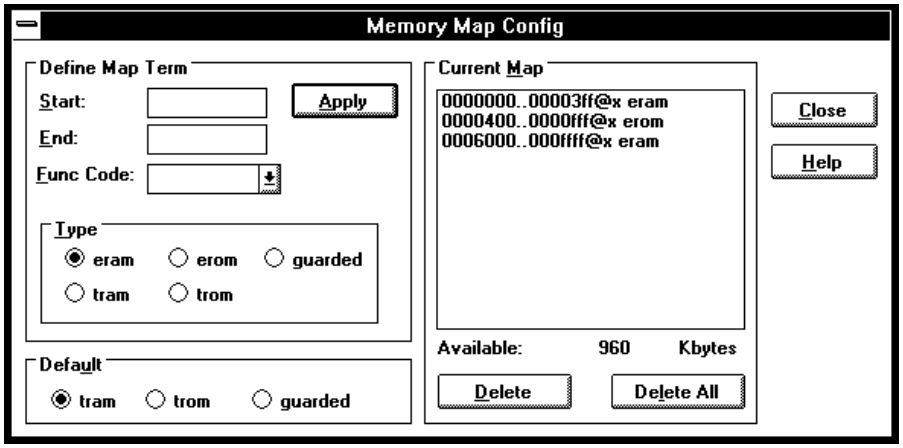
Guarded memory accesses cause emulator execution to break into the monitor program.

Writes to locations mapped as ROM will cause emulator execution to break into the monitor program if these breaks are enabled in the hardware configuration.

Writes to emulation ROM are inhibited. However, even though user program writes to target system memory locations mapped as ROM or guarded memory may result in a break to the monitor, they are not inhibited (that is, the write still occurs).

Memory Map Dialog Box

Choosing the Settings→Emulator Config→Memory Map... (ALT, S, E, M) command opens the following dialog box:



Start	Specifies the starting address of the address range to be mapped.
End	Specifies the end address of the address range to be mapped.
Func Code	Assigns any of the *function codes* to the address range. It's only necessary to specify a function code other than X when mapping overlapping address ranges for different memory spaces. When mapping overlapping ranges, you can only select function codes that haven't already been selected for previously mapped ranges.
Type	Lets you select the memory type of the specified address range.
Apply	Maps the address range specified in the Define Map Term group box.
Default	Specifies whether unmapped memory ranges are target system RAM, target system ROM, or guarded memory.

Current Map	Lists currently mapped ranges.
Available	Indicates the amount of emulation memory available.
Delete	Deletes the address range selected in the Current Map list box.
Delete All	Deletes all of the address ranges in the Current Map list box.
Close	Closes the dialog box.

Command File Command

`MAP addressrange mem_type func_code`
Maps the specified address range with the specified memory type and function code.

`MAP OTH(ER) mem_type`
Specifies the type of the specified non-mapped memory area.

Any of the above command file commands must be preceded and followed by the respective start and end commands:

`MAP STA(RT)`
Starts the memory mapping command section.

`MAP END`
Ends the memory mapping command section.

See Also

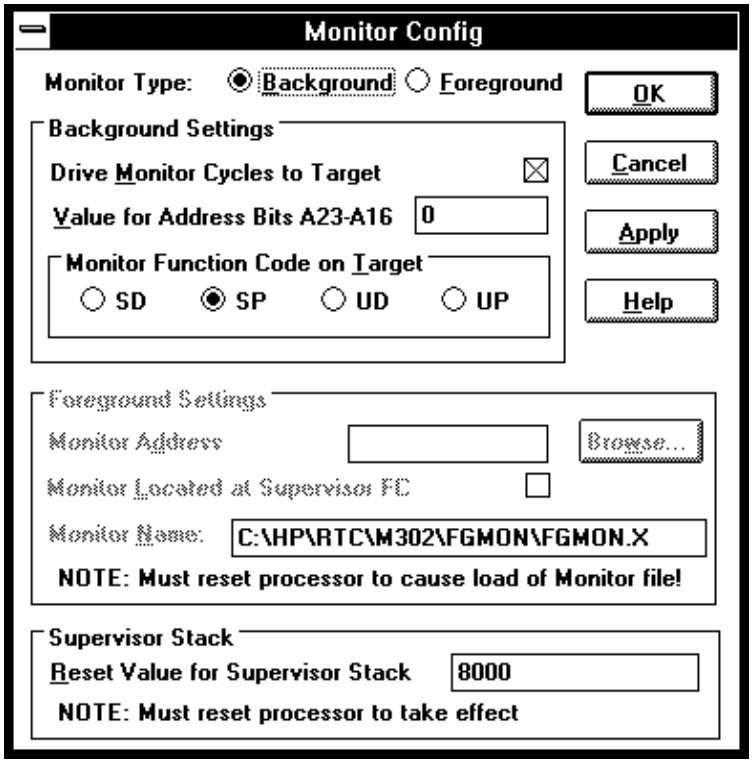
"Mapping Memory" in the "Configuring the Emulator" chapter.

Settings→Emulator Config→Monitor... (ALT, S, E, O)

Selects the type of monitor program and other monitor options.

Monitor Config Dialog Box

Choosing the Settings→Emulator Config→Monitor... (ALT, S, E, O) command opens the following dialog box:



Type	Lets you choose between a background monitor and a foreground monitor.
Reset Value for Supervisor Stack	Specifies the value the stack pointer is set to when the processor transfers from the EMULATION RESET status

to the RUNNING IN MONITOR status. Both types of monitor programs require a stack to be set up in RAM.

Drive Monitor Cycles to Target

Enables or disables the output of background cycles to the target system. The bus cycle contains address, data, and control signals.

Value for Address Bits A23-A16

When background cycles are driven to the target system, this option specifies the value that is output on the upper eight address lines.

Monitor Function Code on Target

When background cycles are driven to the target system, this option specifies the function code that is output. You can select from the following function codes:

SD - Supervisor Data space.
SP - Supervisor Program space.
UD - User Data space.
UP - User Program space.

Monitor Address

Specifies the starting address of the foreground monitor program. The foreground monitor program must be linked at this address.

The starting address must be located on a 2 Kbyte boundary unless you're using the HP 64170 memory board with 1 Mbyte memory modules; in this case, the starting address must be located on an 8 Kbyte boundary.

Monitor Located at Supervisor FC

Specifies that the foreground monitor is located in "Supervisor" memory space. If this option is not selected, no function code is associated with the foreground monitor memory space.

Monitor Name

Lets you enter the name of the foreground monitor object file. The default is C:\HP\RTC\M302\FGMON\FGMON.X (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software). The foreground monitor will be automatically loaded after each Execution→Reset (ALT, E, E) command. Choosing the Apply button does not load the foreground monitor.

Browse...	Opens a file selection dialog box from which you can select the foreground monitor object file to be loaded.
OK	Modifies the monitor configuration as specified and closes the dialog box. When you have selected a foreground monitor, it is not loaded when you choose OK; instead, you must load it using the File→Load Object... (ALT, F, L) command. A foreground monitor will be loaded automatically after each Emulation→Reset (ALT, E, E) command.
Cancel	Cancels the monitor configuration and closes the dialog box.
Apply	Loads the configuration settings into the emulator.

Command File Command

`MON(ITOR) PRO(CESS) BAC(K)`
Selects the background monitor.

`MON(ITOR) PRO(CESS) FOR(E)`
Selects the foreground monitor.

`MON(ITOR) CYC(LE) ENA(BLE)`
Enables the bus cycle output to the target system during background monitor execution.

`MON(ITOR) CYC(LE) DIS(ABLE)`
Disables the bus cycle output to the target system during background monitor execution.

`MON(ITOR) BAD(DRESS) address`
Specifies the upper address lines when background cycles are output to the target system.

`MON(ITOR) FCO(DE) SP/SD/UP/UD`
Assigns SP, SD, UP, or UD function code to the bus cycle output to the target system.

`MON(ITOR) FIL(ENAME) file_name`
Names the foreground monitor object file.

`MON(ITOR) STA(CK) address`
Specifies the value the stack pointer is set to when the emulator breaks into the monitor from the EMULATION RESET status.

`MON(ITOR) LOC(ATE) address`
Specifies the starting address of the monitor.

`MON(ITOR) STA(CK) SUP(ER)/NON(E)`
Assigns Supervisor function code to the monitor.

Any of the above command file commands must be preceded and followed by the respective start and end commands:

`MON(ITOR) STA(RT)`
Starts the monitor option command section.

`MON(ITOR) END`
Ends the monitor option command section.

See Also

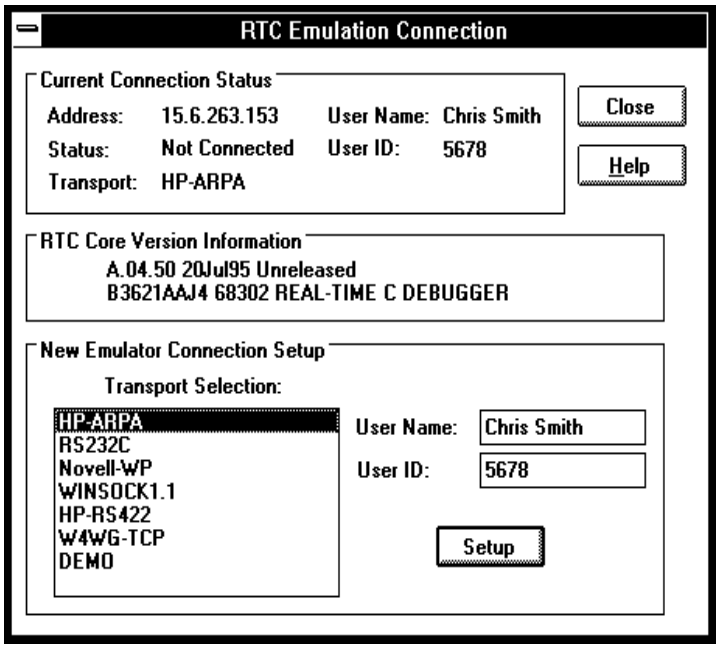
"Selecting the Type of Monitor" in the "Configuring the Emulator" chapter.

Settings→Communication... (ALT, S, C)

Choosing this command opens the RTC Emulation Connection Dialog Box which lets you identify and set up the communication channel between the personal computer and the HP 64700.

RTC Emulation Connection Dialog Box

Choosing the Settings→Communication... (ALT, S, C) command opens the following dialog box:



Current Connection Status

This part of the dialog box shows the current communication settings.

RTC Core Version Information

Displays software version information.

New Emulator Connection Setup

Transport Selection	Lets you choose the type of connection to be made to the HP 64700. Double-clicking causes the current connection to be tried with the given transport. Single-clicking selects the transport for use with the Setup button.
User Name	This name tells the HP 64700 and other users who you are. When other users attempt to access the HP 64700 while you are using it or while it is locked, a message tells them you're using it.
User ID	Another method of identifying yourself to the HP 64700 and other users. This is primarily useful in a mixed UNIX and MS-DOS environment; when a UNIX user tries to unlock an emulator, the user ID is used to look into the /etc/passwd entry on the UNIX host for the user name. If your HP 64700 is on the LAN, we recommend that you change User Name and User ID so that other users can easily tell if an emulator is in use and by whom. Also, if you don't change the User Name/ID from the defaults, the File→Exit HW Locked (ALT, F, H) command has no effect because all users are identical.
Setup	Opens a transport-specific dialog box which usually allows you to change the address and unlock the emulator In the LAN Setup dialog boxes, enter the IP address or network name of the HP 64700. In the RS232C Setup dialog box, select the baud rate and the name of the port (for example, COM1, COM2, etc.) to which the HP 64700 is connected. In the HP-RS422 Setup dialog box, select the baud rate and specify the I/O address you want to use for the HP 64037 card. The I/O address must be a hexadecimal number from 100H through 3F8H, ending in 0 or 8, that does not conflict with other cards in your PC.

	The Connect button in any of these Setup dialog boxes starts the debugger with the specified communication settings.
Close	Either closes the Real-Time C Debugger, if the current connection failed, or simply closes the dialog box.

The Real-Time C Debugger does not allow you to change connection or transport information without leaving the debugger and reentering it. However, any changes you make will be put in the .INI file and take effect the next time you enter the debugger (assuming that you do not override the .INI information on the command line).

The command line options for connection and transport (-E and -T) take precedence over the values in the .lNI file.

Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O)

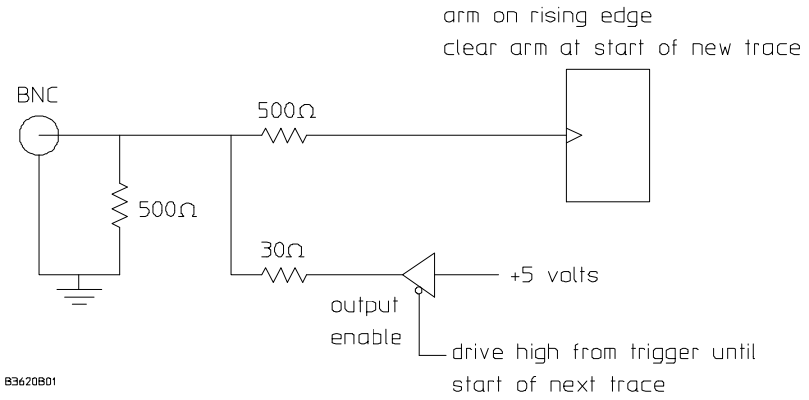
Specifies that the analyzer trigger signal be driven on the BNC port.

Selecting the emulator BNC port for output enables the trigger signals to be fed to external devices (for example, logic analyzers) during tracing.

CAUTION Do not drive the BNC beyond the range of 0 to 5 volts. Doing so may cause permanent damage to the HP 64700.

The BNC's drivers can drive 50 ohm loads.

The following is a logical diagram of the BNC connection. The physical implementation and values of resistors are not exact; this diagram is just to help you understand the BNC interface:



When a trace starts, it stops driving the output (so if nothing else is driving the line, it will fall low due to the 500 ohm pull-down resistor).

When the trigger point is found, the BNC starts driving the output high. It will stay high until the start of the next trace.

Command File Command

```
MOD(E) BNC OUT(PUT_TRIGGER)
```

See Also

"To output the trigger signal on the BNC port" in the "Setting Up the BNC Port" section of the "Configuring the Emulator" chapter.

Settings→BNC→Input to Analyzer Arm (ALT, S, B, I)

Allows the analyzer to receive an arm signal from the BNC port.

This command allows an external trigger signal to be used as an arm (enable) condition for the internal analyzer. The internal analyzer will arm (or enable) on a positive edge TTL signal.

CAUTION Do not drive the BNC beyond the range of 0 to 5 volts. Doing so may cause permanent damage to the HP 64700.

You can use the arm condition when setting up custom trace specifications with the Trace→Find Then Trigger... (ALT, T, D) or Trace→Sequence... (ALT, T, Q) commands. For example, you can trigger on the arm condition or enable the storage of states on the arm condition. The "arm" condition may be selected in "set2" of the Trace Condition or Count Condition dialog boxes.

The BNC port is internally terminated with about 500 ohms; if using a 50 ohm driver, use an external 50 ohm termination (such as the HP 10100C 50 Ohm Feedthrough Termination) to reduce bouncing and possible incorrect triggering.

Command File Command

```
MOD(E) BNC INP(UT_ARM)
```

See Also

Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O) for a logical schematic of the BNC interface.

"To receive an arm condition input on the BNC port" in the "Setting Up the BNC Port" section of the "Configuring the Emulator" chapter.

Settings→Coverage→Coverage ON (ALT, S, V, O)

Selects execution coverage display in the Source window.

This command is not supported when using the HP 64170 emulation memory board.

When the execution coverage display is selected, accessed lines are highlighted in the Source window.

In the coverage calculation, the system counts not only memory access for program code execution but also memory access due to operations such as prefetching.

The coverage calculation must be initialized with the Settings→Coverage→Coverage Reset (ALT, S, V, R) command before the Settings→Coverage→Coverage ON (ALT, S, V, O) command is selected.

The system does not support coverage calculation for target system memory.

Command File Command

```
COV(ERAGE) ON
```

See Also

"To display execution coverage" in the "Making Coverage Measurements" section of the "Debugging Programs" chapter.

Settings→Coverage→Coverage OFF (ALT, S, V, F)

Deselects execution coverage display in the Source window.

This command is not supported when using the HP 64170 emulation memory board.

Command File Command

```
COV(ERAGE) OFF
```

See Also

"To display execution coverage" in the "Making Coverage Measurements" section of the "Debugging Programs" chapter.

Settings→Coverage→Coverage Reset (ALT, S, V, R)

Resets the coverage calculation.

This command is not supported when using the HP 64170 emulation memory board.

The coverage calculation must be initialized with this command before it is started with the Settings→Coverage→Coverage ON (ALT, S, V, O) command. It must also be initialized prior to the coverage calculation immediately after the emulator powerup.

Command File Command

```
COV(ERAGE) RES(ET)
```

See Also

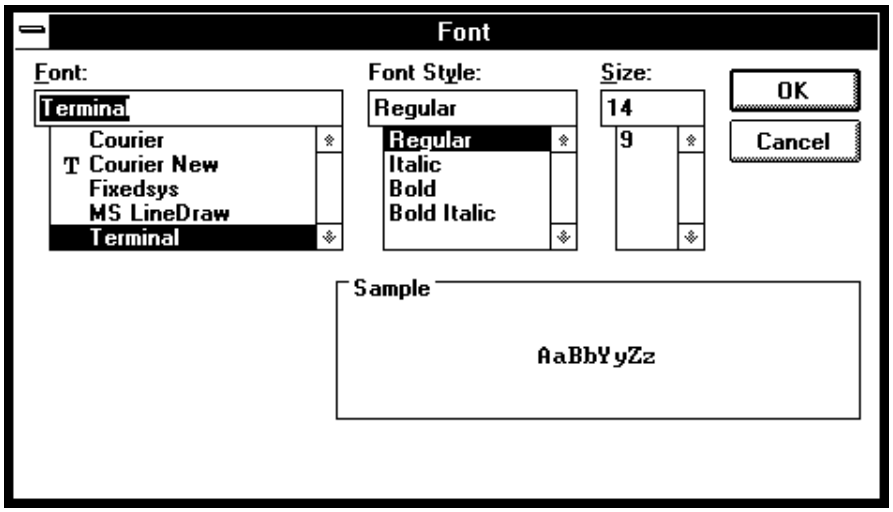
"To display execution coverage" in the "Making Coverage Measurements" section of the "Debugging Programs" chapter.

Settings→Font... (ALT, S, F)

Selects the fonts used in the debugger windows.

Font Dialog Box

Choosing the Settings→Font... (ALT, S, F) command opens the following dialog box:



Font	Lets you select the font to be used in the Real-Time C Debugger interface. The "T" shaped icon indicates a TrueType font.
Font Style	Lets you select the typeface, for example, regular, bold, italic, etc.
Size	Lets you select the size of the characters.
Sample	Shows you what the selected font looks like.
OK	Sets the font, and closes the dialog box.
Cancel	Cancels font setting, and closes the dialog box.

See Also

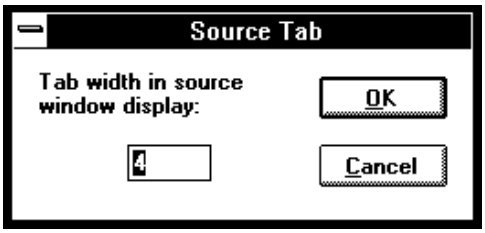
"To change the debugger window fonts" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Settings→Tabstops... (ALT, S, T)

Sets the number of spaces between tab stops.

Source Tab Dialog Box

Choosing the Settings→Tabstops... (ALT, S, T) command opens the following dialog box:



Tab width in source window display	Enter the number of spaces between tab stops. This also affects the tab width for source lines in the Trace window.
OK	Sets the tab stops, and closes the dialog box.
Cancel	Cancels tab stop setting, and closes the dialog box.

See Also

"To set tab stops in the Source window" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O)

Symbol database search is case sensitive.

Command File Command

```
MOD(E) SYM(BOLCASE) ON
```

See Also

Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F)

Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F)

Symbol database search is not case sensitive.

If there are case conflicts (for example, FOO and foo), no warning is given, and you cannot predict which symbol will be used. The symbol that is used depends on what type of symbols FOO and foo are and how they were input by the symbol section of the object file.

Command File Command

```
MOD(E) SYM(BOLCASE) OFF
```

See Also

Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O)

Settings→Extended→Trace Cycles→User (ALT, S, X, T, U)

Traces foreground emulation microprocessor operation.

This is the normal setting.

Command File Command

```
MOD(E) TRA(CECLOCK) USE(R)
```

See Also

Settings→Extended→Trace Cycles→Monitor (ALT, S, X, T, M)
Settings→Extended→Trace Cycles→Both (ALT, S, X, T, B)

Settings→Extended→Trace Cycles→Monitor (ALT, S, X, T, M)

Traces background emulation microprocessor operation.

This is rarely a useful setting when debugging programs.

Command File Command

```
MOD(E) TRA(CECLOCK) BAC(KGROUND)
```

See Also

Settings→Extended→Trace Cycles→User (ALT, S, X, T, U)
Settings→Extended→Trace Cycles→Both (ALT, S, X, T, B)

Settings→Extended→Trace Cycles→Both (ALT, S, X, T, B)

Traces both foreground and background emulation microprocessor operation.

Command File Command

```
MOD(E) TRA(CECLOCK) BOT(H)
```

See Also

Settings→Extended→Trace Cycles→User (ALT, S, X, T, U)
Settings→Extended→Trace Cycles→Monitor (ALT, S, X, T, M)

Settings→Extended→Load Error Abort→ON (ALT, S, X, L, O)

An error during an object file or memory load causes an abort.

Normally, when an error occurs during an object file or memory load, you want the load to stop so that you can fix whatever caused the error.

Command File Command

```
MOD(E) DOW(NLOAD) ERR(ABORT)
```

See Also

Settings→Extended→Load Error Abort→OFF (ALT, S, X, L, F)

Settings→Extended→Load Error Abort→OFF (ALT, S, X, L, F)

An error during an object file or memory load does not cause an abort.

If you expect certain errors during an object file or memory load, for example, if part of the file is located at "guarded" memory or "target ROM," you can choose this command to continue loading in spite of the errors.

Command File Command

```
MOD(E) DOW(NLOAD) NOE(RRABORT)
```

See Also

Settings→Extended→Load Error Abort→ON (ALT, S, X, L, O)

Settings→Extended→Source Path Query→ON (ALT, S, X, S, O)

You are prompted for source file paths.

When the debugger cannot find source file information for the Source or Trace windows, it may prompt you for source file paths depending on the MODE SOURCE setting.

Command File Command

```
MOD(E) SOU(RCE) ASK(PATH)
```

See Also

Settings→Extended→Source Path Query→OFF (ALT, S, X, S, F)

Settings→Extended→Source Path Query→OFF (ALT, S, X, S, F)

You are not prompted for source file paths.

You can turn off source path prompting, for example, to avoid annoying dialog interactions when tracing library functions for which no source files are available.

Command File Command

```
MOD(E) SOU(RCE) NOA(SKPATH)
```

See Also

Settings→Extended→Source Path Query→ON (ALT, S, X, S, O)

Window→Cascade (ALT, W, C)

Arranges, sizes, and overlaps windows.

Windows are sized, evenly, to be as large as possible.

Window→Tile (ALT, W, T)

Arranges and sizes windows so that none are overlapped.

Windows are sized evenly.

Window→Arrange Icons (ALT, W, A)

Rearranges icons in the Real-Time C Debugger window.

Icons are distributed evenly along the lower edge of the Real-Time C Debugger window.

Window→1-9 (ALT, W, 1-9)

Opens the window associated with the number.

The nine most recently opened windows appear in the menu list. If the window you wish to open is not on the list, choose the Window→More Windows... (ALT, W, M) command.

Windows are closed just as are ordinary MS Windows, that is, by opening the control menu and choosing Close or by pressing CTRL+F4.

For details on each of the debugger windows, refer to the "Debugger Windows" section in the "Concepts" information.

Command File Command

`DIS(PLAY) window-name`
Opens the specified window. Use the first three characters of the window name, or, if the window name is "Basic Registers," use "REG."

`ICO(NIC) window-name`
Closes the specified window. Use the first three characters of the window name, or, if the window name is "Basic Registers," use "REG."

See Also

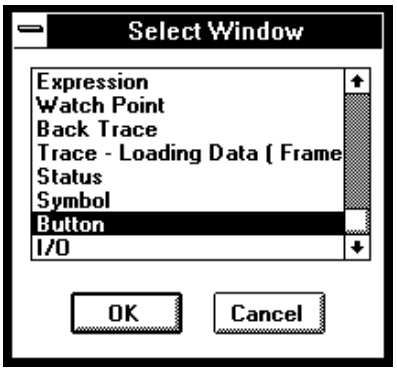
"To open debugger windows" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Window→More Windows... (ALT, W, M)

Presents a list box from which you can select the window to be opened.

Select Window Dialog Box

Choosing the Window→More Windows... (ALT, W, M) command opens the following dialog box:



OK	Opens the window selected in the list box.
Cancel	Closes the dialog box.

Command File Command

`DIS(PLAY) window-name`
Opens the specified window. Use the first three characters of the window name, or, if the window name is "Basic Registers," use "REG."

`ICO(NIC) window-name`
Closes the specified window. Use the first three characters of the window name, or, if the window name is "Basic Registers," use "REG."

See Also

"To open debugger windows" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

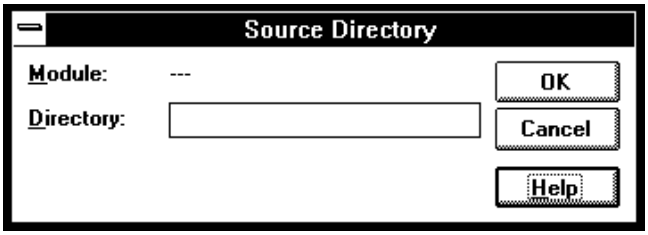
Help→About Debugger/Emulator... (ALT, H, D)

Provides information on the Real-Time C Debugger.

Choosing the Help→About Debugger/Emulator... (ALT, H, D) command opens a dialog box containing the version information on the current Real-Time C Debugger and emulator.

Source Directory Dialog Box

When the source file associated with a symbol cannot be found in the current directory, the following dialog box is opened:



Module	Shows the symbol whose source file could not be found.
Directory	Lets you enter the directory in which the source file associated with the symbol may be found.
OK	Adds the directory entered in the Directory text box to the source file search path.
Cancel	Closes the dialog box.

WAIT Command Dialog Box

This dialog box appears when the WAIT command is included in a command file, break macro, or button.

Choosing the STOP button cancels the WAIT command.

9

Window Control Menu Commands

Window Control Menu Commands

This chapter describes the commands that can be chosen from the *control menus* in debugger windows.

- Common Control Menu Commands
- Button Window Commands
- Device Regs Window Commands
- Expression Window Commands
- I/O Window Commands
- Memory Window Commands
- Register Window Commands
- Source Window Commands
- Symbol Window Commands
- Trace Window Commands
- WatchPoint Window Commands

Common Control Menu Commands

This section describes commands that appear in the control menus of most of the debugger windows:

- Copy→Window (ALT, -, P, W)
- Copy→Destination... (ALT, -, P, D)

Copy→Window (ALT, -, P, W)

Copies the current window contents to the destination file specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
COP(Y) BAC(KTRACE)

COP(Y) BUT(TON)

COP(Y) EXP(RESSION)

COP(Y) IO

COP(Y) MEM(ORY)

COP(Y) REG(ISTER)

COP(Y) SOU(RCE)

COP(Y) WAT(CHPOINT)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Copy→Destination... (ALT, -, P, D)

Names the listing file to which debugger information may be copied.

This command opens a file selection dialog box from which you can select the listing file. Listing files have the extension ".LST".

Command File Command

```
COP(Y) TO filename
```

See Also

"To change the list file destination" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Button Window Commands

This section describes the following command:

- Edit... (ALT, -, E)

Edit... (ALT, -, E)

Lets you define and label buttons in the Button window.

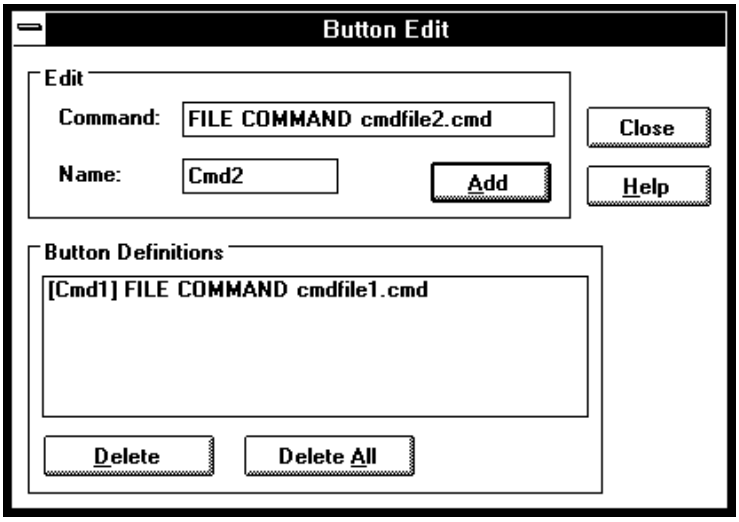
You can set up buttons to execute commonly used commands or command files.

Note that the Copy→Window command will generate a listing file that contains a header followed by commands needed to recreate the buttons. By removing the header, this file may be used as a command file.

Alternatively, you can log commands to a command file as you edit the buttons (refer to To create a command file in the "Using Command Files" section of the "Using the Debugger Interface" chapter). To recreate the buttons, just run the command file that you created while editing the buttons.

Button Edit Dialog Box

Choosing the Edit... (ALT, -, E) command opens the following dialog box:



Command	Specifies the command to be associated with the button. Command syntax is described at the bottom of most help topics under the "Command File Command" heading. Also, look in the "Command File and Macro Command Summary" chapter in the "Reference" part. You can only enter a single command here; if you want a series of commands to be executed when this button is used, put them in a command file and use the command "FILE COMMAND filename," where "filename" is the name of your command file.
Name	Specifies the button label to be associated with the command.
Add	Adds the button to the button window.
Button Definitions	Lists the currently defined buttons. You can select button definitions for deletion by clicking on them.

Delete	Deletes the button definition selected in the Button Definitions list box.
Delete All	Deletes all buttons from the Button window.
Close	Closes the dialog box.

Command File Command

```
BUTTON label "command"
```

See Also

"To create buttons that execute command files" in the "Using Command Files" section of the "Using the Debugger Interface" chapter.

Device Regs Window Commands

This section describes the following command:

- Continuous Update (ALT, -, U)

Continuous Update (ALT, -, U)

Specifies whether the Device Regs window contents should be continuously updated while running programs.

A check mark (√) next to the command shows that continuous update is active.

Expression Window Commands

This section describes the following commands:

- Clear (ALT, -, R)
- Evaluate... (ALT, -, E)

Clear (ALT, -, R)

Erases the contents of the Expression window.

Command File Command

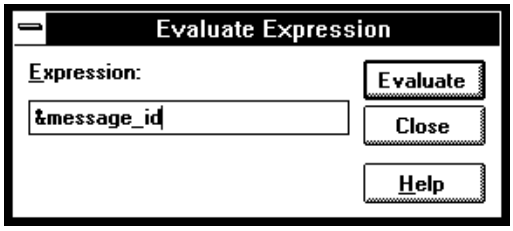
```
EVA(LUATE) CLE(AR)
```

Evaluate... (ALT, -, E)

Evaluates expressions and displays the results in the Expression window.

Evaluate Expression Dialog Box

Choosing the Evaluate... (ALT, -, E) command opens the following dialog box:



Expression	Lets you enter the expression to be evaluated.
Evaluate	Makes the evaluation and places the results in the Expression window.
Close	Closes the dialog box.

Command File Command

```
EVA(LUATE) address

EVA(LUATE) "strings"
```

See Also

"Symbols" in the "Expressions in Commands" chapter.

I/O Window Commands

This section describes the following command:

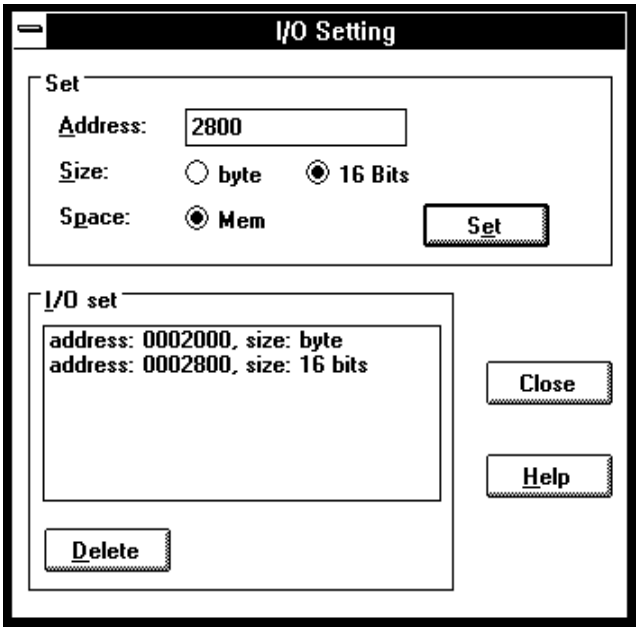
Define... (ALT, -, D)

Define... (ALT, -, D)

Adds or deletes memory mapped I/O locations from the I/O window.

I/O Setting Dialog Box

Choosing the Edit→Definition... command opens the following dialog box:



Address	Specifies the address of the I/O location to be defined.
Size	Specifies the data format of the I/O location to be defined. You can select the Byte or 16 Bits option.
Space	Specifies whether the I/O location is in memory or I/O space.
Set	Adds the specified I/O location.
I/O set	Displays the information on the I/O locations that have been set.
Delete	Deletes the I/O locations selected in the I/O set list box.
Close	Closes the dialog box.

Command File Command

`IO BYTE/WORD/LONG IOSPACE/MEMORY address TO data`
Replaces the contents of the specified I/O address with the specified value in the specified size.

`IO SET BYTE/WORD/LONG IOSPACE/MEMORY address`
Registers the I/O address to be displayed in the specified size.

`IO DEL(ETE) BYTE/WORD/LONG IOSPACE/MEMORY address`
Deletes the I/O specified with its address and size.

See Also

"Displaying and Editing I/O Locations" in the "Debugging Programs" chapter.

Memory Window Commands

This section describes the following commands:

- Display→Linear (ALT, -, D, L)
- Display→Block (ALT, -, D, B)
- Display→Byte (ALT, -, D, Y)
- Display→16 Bits (ALT, -, D, 1)
- Display→32 Bits (ALT, -, D, 3)
- Search... (ALT, -, R)
- Utilities→Copy... (ALT, -, U, C)
- Utilities→Fill... (ALT, -, U, F)
- Utilities→Image... (ALT, -, U, I)
- Utilities→Load... (ALT, -, U, L)
- Utilities→Store... (ALT, -, U, S)

Display→Linear (ALT, -, D, L)

Displays memory contents in single column format.

Command File Command

```
MEM(ORY) ABS(OLUTE)
```

Display→Block (ALT, -, D, B)

Displays memory contents in multicolumn format.

Command File Command

```
MEM(ORY) BLO(CK)
```

Display→Byte (ALT, -, D, Y)

Displays memory contents as bytes.

Command File Command

```
MEM(ORY) BYTE
```

Display→16 Bit (ALT, -, D, 1)

Displays memory contents as 16-bit values.

Command File Command

```
MEM(ORY) WORD
```

Display→32 Bit (ALT, -, D, 3)

Displays memory contents as 32-bit values.

Command File Command

```
MEM(ORY) LONG
```

Search... (ALT, -, R)

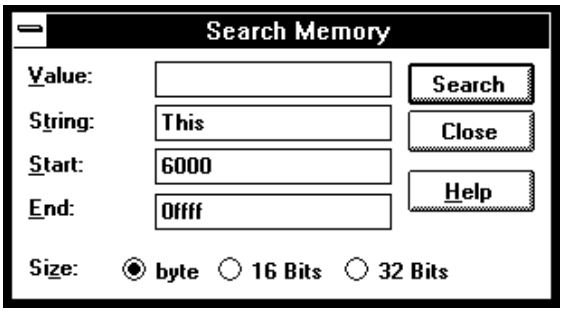
Searches for a value or string in a range of memory.

When the value or string is found, the location is displayed in the Memory window. Choose the Window→Memory command to open the window.

The value or string can be selected from another window (in other words, copied to the clipboard) before choosing the command; the contents of the clipboard will automatically appear in the dialog box that is opened.

Search Memory Dialog Box

Choosing the Search... (ALT, -, R) command opens the following dialog box:



Value	Lets you enter a value.
String	Lets you enter a string.
Start	Lets you enter the starting address of the memory range to search.
End	Lets you enter the end address of the memory range to search.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Searches for the specified value or string.

Close	Closes the dialog box.

Command File Command

```
SEA(RCH) MEM(ORY) BYTE/WORD/LONG addr_range value

SEA(RCH) MEM(ORY) STR(ING) "string"
```

See Also

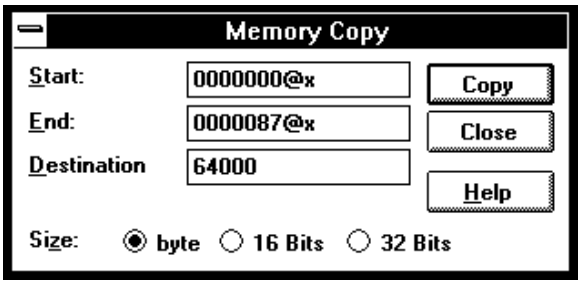
"To search memory for a value or string" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Copy... (ALT, -, U, C)

Copies the contents of one memory area to another.

Memory Copy Dialog Box

Choosing the Utilities→Copy... (ALT, -, U, C) command opens the following dialog box:



Start	Lets you enter the starting address of the source memory area.
End	Lets you enter the end address of the source memory area.
Destination	Specifies the starting address of the destination memory area.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Copies the memory contents.
Close	Closes the dialog box.

Command File Command

```
MEM(ORY) COP(Y) size address_range address
```

See Also

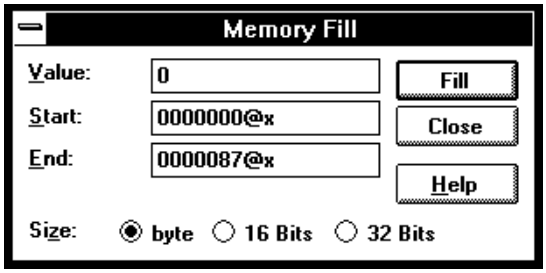
"To copy memory to a different location" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Fill... (ALT, -, U, F)

Fills a range of memory with a specified value.

Memory Fill Dialog Box

Choosing the Utilities→Fill... (ALT, -, U, F) command opens the following dialog box:



Value	Lets you enter the filling value.
Start	Lets you enter the starting address of the memory area to be filled.
End	Lets you enter the end address of the memory area to be filled.
Size	Selects the size of the filling value. If the value specified is larger than can fit in the size selected, the upper bits of the value are ignored. You can select the size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Executes the command.

Close Closes the dialog box.

Command File Command

```
MEM(ORY) FIL(L) size address_range data
```

See Also

"To modify a range of memory with a value" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Image... (ALT, -, U, I)

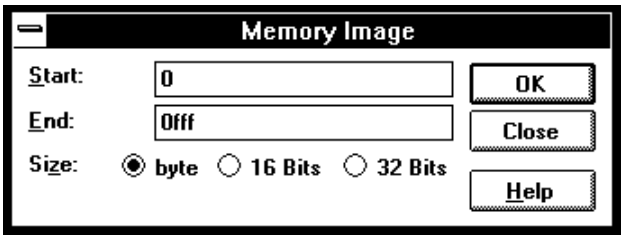
Copies the contents of a target system memory range into the corresponding emulation memory range.

You can copy programs that are in target system ROM to emulation memory. Once the program code is in emulation memory, you can use features like breakpoints, run until, etc.

The address range must be mapped as emulation memory before choosing this command.

Memory Image Dialog Box

Choosing the Utilities→Image... (ALT, -, U, I) command opens the following dialog box:



Start	Lets you enter the starting address of the memory area.
End	Lets you enter end address of the memory area.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Copies the target system memory into emulation memory.
Close	Closes the dialog box.

Command File Command

```
MEM(ORY) IMA(GE) size address_range
```

See Also

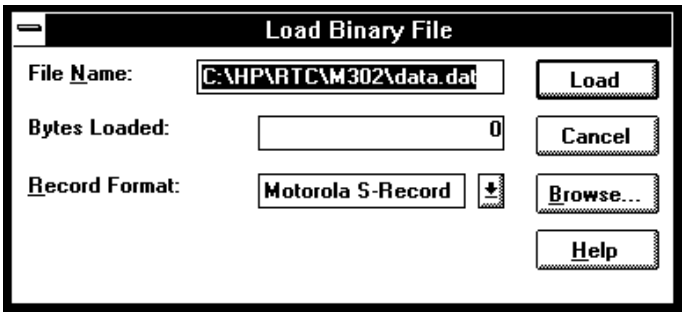
"To copy target system memory into emulation memory" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Load... (ALT, -, U, L)

Loads memory contents from a previously stored file.

Load Binary File Dialog Box

Choosing the Utilities→Load... (ALT, -, U, L) command opens the following dialog box:



File Name	Lets you enter the name of the file to load memory from.
Bytes Loaded	After you choose the Import button, this box shows the number of bytes that are loaded.
Record Format	Lets you specify the format of the file from which you're loading memory. You can load Motorola S-Record or Intel Hexadecimal format files.
Load	Starts the memory load.
Cancel	Closes the dialog box.
Browse...	Opens a file selection dialog box from which you can select the file name.

Command File Command

```
MEM(ORY) LOA(D) MOT(OSREC) filename

MEM(ORY) LOA(D) INT(ELHEX) filename
```

See Also

"To copy target system memory into emulation memory" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

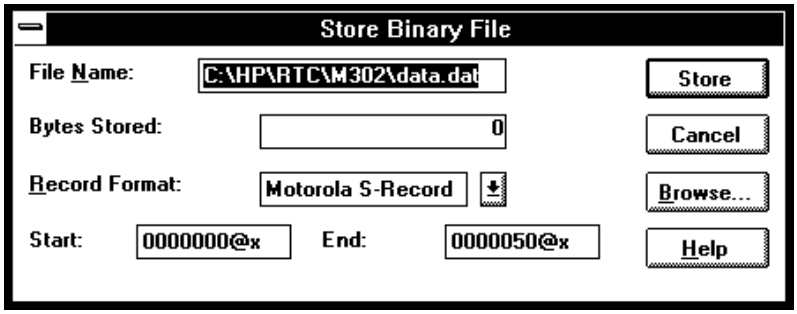
Utilities→Store... (ALT, -, U, S)

Utilities→Store... (ALT, -, U, S)

Stores memory contents to a binary file.

Store Binary File Dialog Box

Choosing the Utilities→Store... (ALT, -, U, S) command opens the following dialog box:



File Name	Lets you enter the name of the file to which memory contents are stored.
Bytes Stored	After you choose the Export button, this box shows the number of bytes that are stored.
Record Format	Lets you specify the format of the file to which you're storing memory. You can select Motorola S-Record or Intel Hexadecimal formats.

Start	Lets you enter the starting address of the memory range to be stored.
End	Lets you enter the ending address of the memory range to be stored.
Store	Starts the memory store.
Cancel	Closes the dialog box.
Browse...	Opens a file selection dialog box from which you can select a file name.

Command File Command

```
MEM(ORY) STO(RE) MOT(OSREC) addr-range filename
```

```
MEM(ORY) STO(RE) INT(ELHEX) addr-range filename
```

See Also

"To copy target system memory into emulation memory" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Load... (ALT, -, U, L)

Register Window Commands

This section describes the following command:

Copy→Registers (ALT, -, P, R)

Copy→Registers (ALT, -, P, R)

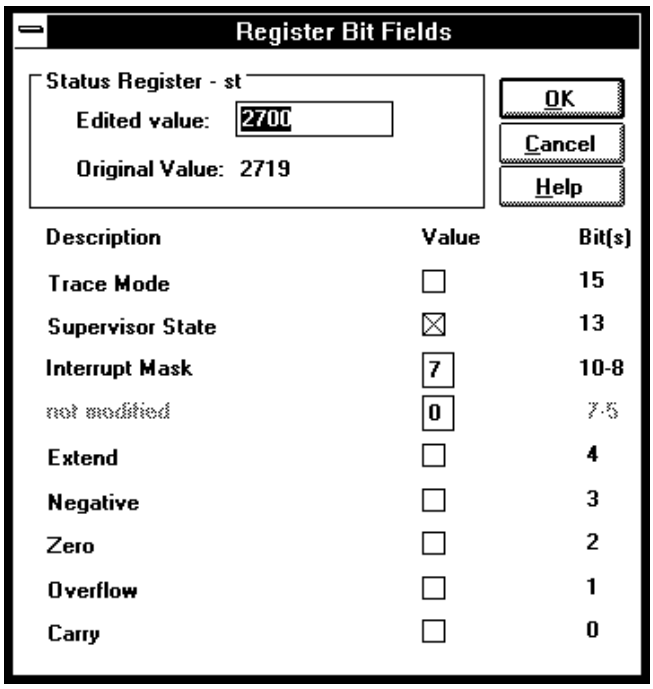
Copies the current Register window contents to the destination file specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
COP(Y) REG(ISTER)
```

Register Bit Fields Dialog Box

When a register has bit-fields, a dialog will pop-up and the register value may be edited by changing the whole value or by editing individual bit-fields.



When editing in the dialog box, a carriage-return is the same as choosing the OK button. To end an edit of a field within the dialog box without quitting, use the Tab key.

Edited Value — Shows the register value that corresponds to the selections made below. You can also change the register's value by modifying the value in this text box.

Original Value — Shows the value of the register when the dialog box was opened. If the register could not be read, 'XXXXXXXX' is displayed.

OK	Modifies the register as specified, and closes the dialog box.
Cancel	Closes the dialog box without modifying the register.

Source Window Commands

This section describes the following commands:

- Display→Mixed Mode (ALT, -, D, M)
- Display→Source Only (ALT, -, D, S)
- Display→Select Source... (ALT, -, D, L)
- Search→String... (ALT, -, R, S)
- Search→Function... (ALT, -, R, F)
- Search→Address... (ALT, -, R, A)
- Search→Current PC (ALT, -, R, C)

Display→Mixed Mode (ALT, -, D, M)

Chooses the source/mnemonic mixed display mode.

Command File Command

```
MOD(E) MNE(MONIC) ON
```

See Also

"To display source code mixed with assembly instructions" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Display→Source Only (ALT, -, D, S)

Chooses the source only display mode.

Command File Command

```
MOD(E) MNE(MONIC) OFF
```

See Also

"To display source code only" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

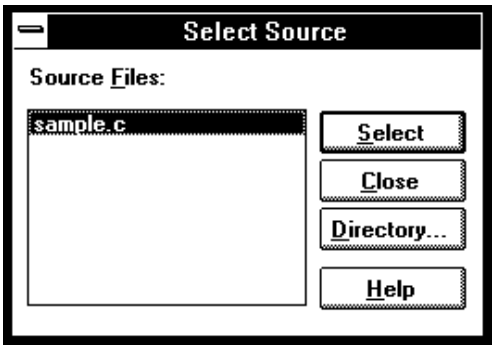
Display→Select Source... (ALT, -, D, L)

Displays the contents of the specified C source file in the Source window.

This command is disabled before the object file is loaded or when no source is available for the loaded object file.

Select Source Dialog Box

Choosing the Display→Select Source... (ALT, -, D, L) command opens the following dialog box:



Source Files	Lists C source files associated with the loaded object file. You can select the source file to be displayed from this list.
Select	Switches the Source window contents to the selected source file.
Close	Closes the dialog box.
Directory	Opens the Search Directories Dialog Box from which you can add directories to the search path.

Command File Command

```
FIL(E) SOU(RCE) module_name
```

See Also

"To display source files by their names" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Search→String... (ALT, -, R, S)

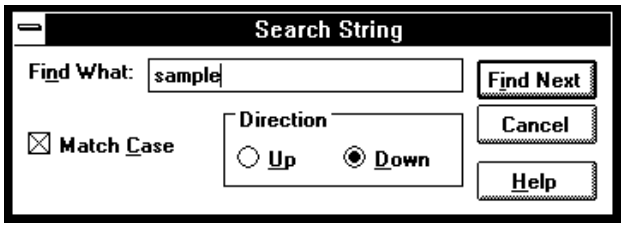
Searches for, and displays, a string in the Source window.

The search starts from the current cursor position in the Source window, may be either forward or backward, and may be case sensitive.

The string can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

Search String Dialog Box

Choosing the Search→String... (ALT, -, R, S) command opens the following dialog box:



Find What	Lets you enter the string.
Match Case	Selects or deselects case matching.
Up	Specifies that the search be from the current cursor position backward.
Down	Specifies that the search be from the current cursor position forward.
Find Next	Searches for the string.
Close	Closes the dialog box.

Command File Command

`SEA(RCH) STR(ING) FOR/BACK ON/OFF strings`
Searches the specified string in the specified direction with the case matching option ON or OFF.

See Also

"To search for strings in the source files" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Search→Function... (ALT, -, R, F)

Searches for, and displays, a function in the Source window.

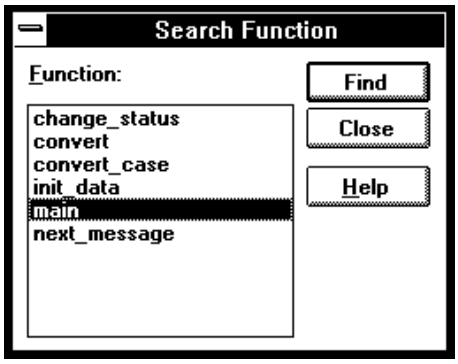
The object file and symbols must be loaded before you can choose this command.

Note

This command displays the source file based on the function information in the object file. Depending on the structure of the function, the command may fail in displaying the declaration of the function.

Search Function Dialog Box

Choosing the Search→Function... (ALT, -, R, F) command opens the following dialog box:



Function	Lets you select the function to search for.
Find	Searches the specified function.
Close	Closes the dialog box.

Command File Command

```
SEA(RCH) FUNC(TION) func_name
```

See Also

"To search for function names in the source files" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

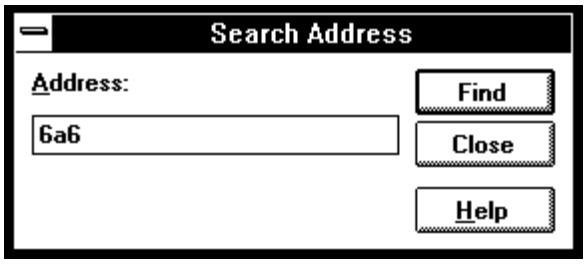
Search→Address... (ALT, -, R, A)

Searches for, and displays, an address in the Source window.

Address expressions such as function names or symbols can be selected from another window (in other words, copied to the clipboard) before choosing the command; the contents of the clipboard will automatically appear in the dialog box that is opened.

Search Address Dialog Box

Choosing the Search→Address... (ALT, -, R, A) command opens the following dialog box:



Address	Lets you enter the address to search for.
Find	Searches for the specified address.
Close	Closes the dialog box.

Command File Command

`CUR(SOR) address`
When used before the COME command, this command can be used to run to a particular address.

See Also

"To search for addresses in the source files" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Search→Current PC (ALT, -, R, C)

Searches for, and displays, the location of the current program counter in the Source window.

Command File Command

`CUR(SOR) PC`
This command can be used to show the current PC in the Source window.

Search Directories Dialog Box

Choosing the Directories... button in the Select Source dialog box opens the following dialog box:

Directory	Lets you enter the directory to be added to the source file search path.
Search Source Directories	Lists the directories in the source file search path.
Add	Adds the directory entered in the Directory text box to the source file search path.
Delete	Deletes the directory in the Directory text box from the source file search path.
Close	Closes the dialog box.

See Also

"To specify source file directories" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Symbol Window Commands

This section describes the following commands:

- Display→Modules (ALT, -, D, M)
- Display→Functions (ALT, -, D, F)
- Display→Externals (ALT, -, D, E)
- Display→Locals... (ALT, -, D, L)
- Display→Asm Globals (ALT, -, D, G)
- Display→Asm Locals... (ALT, -, D, A)
- Display→User defined (ALT, -, D, U)
- Copy→Window (ALT, -, P, W)
- Copy→All (ALT, -, P, A)
- FindString→String... (ALT, -, D, M)
- User defined→Add... (ALT, -, U, A)
- User defined→Delete (ALT, -, U, D)
- User defined→Delete All (ALT, -, U, L)

Display→Modules (ALT, -, D, M)

Displays the symbolic module information from the loaded object file.

Command File Command

```
SYM(BOL) LIS(T) MOD(ULE)
```

See Also

"To display program module information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Functions (ALT, -, D, F)

Displays the symbolic function information from the loaded object file.

The Symbol window displays the name, type and address range for C functions.

Command File Command

```
SYM(BOL) LIS(T) FUN(CTION)
```

See Also

"To display function information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Externals (ALT, -, D, E)

Displays the global variable information from the loaded object file.

The Symbol window displays the name, type and address for global variables.

Command File Command

```
SYM(BOL) LIS(T) EXT(ERNAL)
```

See Also

"To display external symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Locals... (ALT, -, D, L)

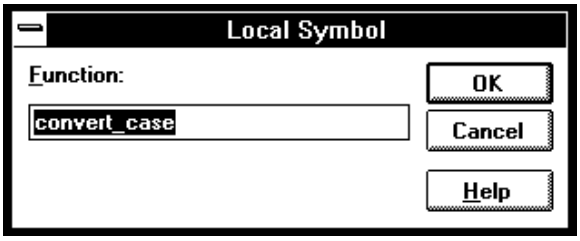
Displays the local variable information on the specified function.

The function name can be selected from another window (in other words, copied to the clipboard) before choosing the command; the clipboard contents automatically appear in the dialog box that is opened.

The Symbol window displays the name, type and offset from the frame pointer for the local variables for the specified function.

Local Symbol Dialog Box

Choosing the Display→Locals... (ALT, -, D, L) command opens the following dialog box:



Function	Selects the function for which the local variable information is displayed.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) LIS(T) INT(ERNAL) function
```

See Also

"To display local symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Asm Globals (ALT, -, D, G)

Displays the global Assembler symbol information from the loaded object file.

The Symbol window displays the name and address for the global assembler symbols.

Command File Command

```
SYM(BOL) LIS(T) GLO(BALS)
```

See Also

"To display global assembler symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Asm Locals... (ALT, -, D, A)

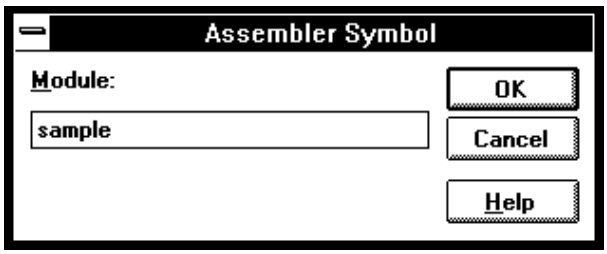
Displays the local symbol information from the specified module.

The module name can be selected from another window (in other words, copied to the clipboard) before choosing the command; the clipboard contents automatically appear in the dialog box that is opened.

The Symbol window displays the name and address for the local symbols for the specified module.

Assembler Symbol Dialog Box

Choosing the Display→Asm Locals... (ALT, -, D, A) command opens the following dialog box:



Module	Selects the module for which the local symbols are displayed.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) LIS(T) LOC(AL) module
```

See Also

"To display local assembler symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→User defined (ALT, -, D, U)

Displays the user-defined symbol information.

The Symbol window displays the name and address for the user-defined symbols.

The User defined→Add... (ALT, -, D, U) command adds the user-defined symbols.

Command File Command

```
SYM(BOL) LIS(T) USE(R)
```

See Also

"To display user-defined symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Copy→Window (ALT, -, P, W)

Copies the information currently displayed in the Symbol window to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
SYM(BOL) COP(Y) DIS(PLAY)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Copy→All (ALT, -, P, A)

Copies all the symbol information to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
SYM(BOL) COP(Y) ALL
```

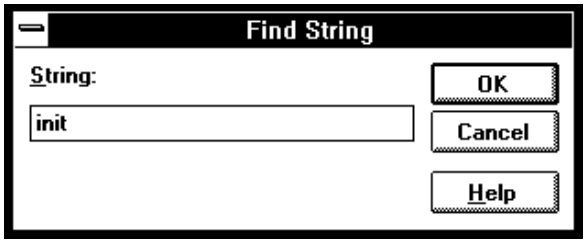
FindString→String... (ALT, -, F, S)

Displays the symbols that contain the specified string.

This command performs a case-sensitive search.

Symbol Matches Dialog Box

Choosing the FindString→String... (ALT, -, F, S) command opens the following dialog box:



String	Specifies the string.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) MAT(CH) string
```

See Also

"To display the symbols containing the specified string" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

User defined→Add... (ALT, -, U, A)

Adds the specified user-defined symbol.

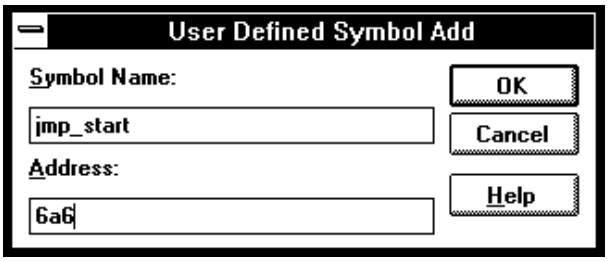
User-defined symbols may be used in debugger commands just like other program symbols.

The symbol name must satisfy the following requirements:

- The name must begin with an alphabetical, _ (underscore), or ? character.
- The following characters must be any of alphanumerical, _ (underscore), or ? characters.
- The maximum number of characters is 256.

User defined Symbol Dialog Box

Choosing the User defined→Add... (ALT, -, U, A) command opens the following dialog box:



Symbol Name	Specifies the symbol to be added.
Address	Specifies the address of the symbol.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) ADD symbol_nam address
```

See Also

"To create a user-defined symbol" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

User defined→Delete (ALT, -, U, D)

Deletes the specified user-defined symbol.

This command deletes the user-defined symbol selected in the Symbol window.

Command File Command

```
SYM(BOL) DEL(ETE) symbol_nam
```

See Also

"To delete a user-defined symbol" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

User defined→Delete All (ALT, -, U, L)

Deletes all the user-defined symbols.

Command File Command

```
SYM(BOL) DEL(ETE) ALL
```

Trace Window Commands

This section describes the following commands:

- Display→Mixed Mode (ALT, -, D, M)
- Display→Source Only (ALT, -, D, S)
- Display→Bus Cycle Only (ALT, -, D, C)
- Display→Count→Absolute (ALT, -, D, C, A)
- Display→Count→Relative (ALT, -, D, C, R)
- Copy→Window (ALT, -, P, W)
- Copy→All (ALT, -, P, A)
- Search→Trigger (ALT, -, R, T)
- Search→State... (ALT, -, R, S)
- Trace Spec Copy→Specification (ALT, -, T, S)
- Trace Spec Copy→Destination... (ALT, -, T, D)

Display→Mixed Mode (ALT, -, D, M)

Chooses the source/mnemonic mixed display mode.

Command File Command

```
TRA(CE) DIS(PLAY) MIX(ED)
```

See Also

"To display source code mixed with assembly instructions" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Display→Source Only (ALT, -, D, S)

Selects the source only display mode.

Command File Command

```
TRA(CE) DIS(PLAY) SOU(RCE)
```

See Also

"To display bus cycles" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Bus Cycle Only (ALT, -, D, C)

Selects the bus cycle only display mode.

Command File Command

```
TRA(CE) DIS(PLAY) BUS
```

See Also

To display bus cycles in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Count→Absolute (ALT, -, D, C, A)

Selects the absolute mode (the total time elapsed since the trigger) for count information.

Command File Command

```
TRA(CE) DIS(PLAY) ABS(OLUTE)
```

See Also

"To display absolute or relative counts" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Count→Relative (ALT, -, D, C, R)

Selects the relative mode (the time interval between the current and previous cycle) for count information.

Command File Command

```
TRA(CE) DIS(PLAY) REL(ATIVE)
```

See Also

"To display absolute or relative counts" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Copy→Window (ALT, -, P, W)

Copies the information currently in the Trace window to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
TRA(CE) COP(Y) DIS(PLAY)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Copy→All (ALT, -, P, A)

Copies all the trace information to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
TRA(CE) COP(Y) ALL
```

Search→Trigger (ALT, -, R, T)

Positions the trigger state at the top of the Trace window.

Command File Command

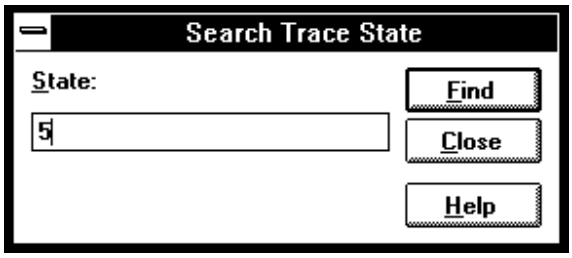
```
TRA(CE) FIN(D) TRI(GGER)
```

Search→State... (ALT, -, R, S)

Positions the specified state at the top of the Trace window.

Search Trace State Dialog Box

Choosing the Search→State... (ALT, -, R, S) command opens the following dialog box:



State	Lets you enter the trace state number to search for.
Find	Searches for the specified trace state.
Close	Closes the dialog box.

Command File Command

```
TRA(CE) FIN(D) STA(TE) state_num
```

Trace Spec Copy→Specification (ALT, -, T, S)

Copies the current trace specification to the listing file.

Command File Command

```
TRA(CE) COP(Y) SPE(C)
```

Trace Spec Copy→Destination... (ALT, -, T, D)

Names the listing file to which debugger information may be copied.

This command opens a file selection dialog box from which you can select the listing file. Listing files have the extension ".LST".

Command File Command

```
COP(Y) TO filename
```

WatchPoint Window Commands

This section describes the following command:

Edit...

Edit... (ALT, -, E)

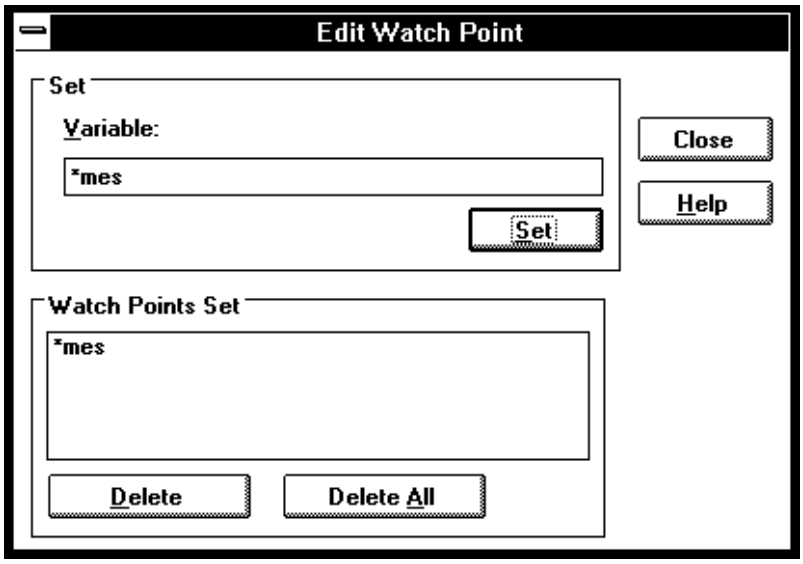
Registers or deletes watchpoints.

Variables can be selected from the another window (in other words, copied to the clipboard) before choosing the Edit... (ALT, -, E) command from the WatchPoint window's control menu, and they will automatically appear in the dialog box that is opened.

Dynamic variables can be registered and displayed in the WatchPoint window when the current program counter is in the function in which the variable is declared. If the current program counter is not in the function, the variable name is invalid and results in an error.

WatchPoint Dialog Box

Choosing the Edit... (ALT, -, E) command from the WatchPoint window's control menu opens the following dialog box:



Variable	Lets you enter the name of the variable to be registered as a watchpoint. The contents of the clipboard, usually a variable selected from the another window, automatically appears in this text box.
Watch Points Set	Lists the current watchpoints and allows you to select the watchpoint to be deleted.
Set	Copies the specified variable to the WatchPoint window.
Delete	Deletes the variable selected in the Watch Points Set box.
Delete All	Deletes all the watchpoints.
Close	Closes the dialog box.

Command File Command

`WP SET address`
Registers the specified address as a watchpoint.

`WP DEL(ETE) address`
Deletes the specified watchpoint.

`WP DEL(ETE) ALL`
Deletes all the current watchpoints.

See Also

"To monitor a variable in the WatchPoint window" in the "Displaying and Editing Variables" section of the "Debugging Programs" chapter.

"Symbols" in the "Expressions in Commands" chapter.

10

Window Pop-Up Commands

Window Pop-Up Commands

This chapter describes the commands that can be chosen from the pop-up menus in debugger windows. Pop-Up menus are accessed by clicking the right mouse button in the window.

- BackTrace Window Pop-Up Commands
- Source Window Pop-Up Commands

BackTrace Window Pop-Up Commands

- Source at Stack Level

Source at Stack Level

For the cursor-selected function in the BackTrace window, this command displays the function call in the Source window.

Source Window Pop-Up Commands

- Set Breakpoint
- Clear Breakpoint
- Evaluate It
- Add to Watch
- Run to Cursor

Set Breakpoint

Sets a breakpoint on the line containing the cursor. Refer to the Breakpoint→Set at Cursor (ALT, B, S) command.

Clear Breakpoint

Deletes the breakpoint on the line containing the cursor. Refer to the Breakpoint→Delete at Cursor (ALT, B, D) command.

Evaluate It

Evaluates the clipboard contents and places the result in the Expression window. Refer to the Evaluate... (ALT, -, E) command available from the Expression window's control menu.

Add to Watch

Adds the selected variable (that is, the variable copied to the clipboard) to the WatchPoint window. Refer to the Variable→Edit... (ALT, V, E) command.

Run to Cursor

Executes the program up to the Source window line containing the cursor. Refer to the Execution→Run to Cursor (ALT, R C) command.

11

Other Command File and Macro Commands

Other Command File and Macro Commands

This chapter describes the commands that are only available in command files, break macros, or buttons.

- BEEP
- EXIT
- FILE CHAINCMD
- FILE RERUN
- NOP
- TERMCOM
- WAIT

BEEP

Sounds beep during command file or break macro execution.

Command File Command

BEEP

EXIT

Exits, or conditionally exits, command file execution.

Command File Command

`EXIT`
Exits command file execution.

`EXIT VAR(IABLE) address value`
Exits command file execution if the variable contains the value.

`EXIT REG(ISTER) regname value`
Exits command file execution if the register contains the value.

`EXIT MEM(ORY) BYTE/WORD/LONG address value`
Exits command file execution if the memory location contains the value.

`EXIT IO BYTE/WORD address value`
Exits command file execution if the I/O location contains the value.

FILE CHAINCMD

Chains command file execution.

This command lets you run one command file from another nonrecursively; in other words, control is not returned to the original command file.

By contrast, the FILE COMMAND command is recursive; if you use the FILE COMMAND command to run one command file from another, control will be returned to the original command file. FILE COMMAND commands can be nested four levels deep.

Command File Command

```
FILE CHAINCMD filename
```

FILE RERUN

Starts command file execution over again.

This command is useful for looping stimulus files or running a demo or other command file continuously.

Command File Command

```
FILE RERUN
```

NOP

No operation.

This command may be used to prefix comment lines in command files.

Command File Command

```
NOP
```

```
NOP comments
```

TERMCOM

Sends Terminal Interface commands to the HP 64700.

The HP 64700 Card Cage contains a low-level Terminal Interface, which allows you to control the emulator's functions directly. You can use the TERMCOM command to bypass the RTC Interface and send commands directly to the low-level Terminal Interface.

There is no window in the RTC Interface where you can execute TERMCOM commands directly. The only way to execute them with the RTC Interface is to make them part of a command file and then run the command file from an RTC Interface window.

You may need to start a unique target system that requires emulator intervention that is only available through the Terminal Interface. You can create the command file and then execute it at the appropriate time using a command such as File→Run Cmd File..., and place the name of your command file in the Run Command File dialog box.

The danger in using Terminal Interface commands via the TERMCOM command is that the RTC Interface may not be updated to know the state of the emulator. Some Terminal Interface commands can be executed by using the TERMCOM command, and the RTC Interface will not know that they were executed. Other Terminal Interface commands can be executed and the RTC Interface will be updated immediately. For example:

- If you have a command in your command file that changes the setting of RealTime→Monitor Intrusion→Disallowed/Allowed, (such as, TERMCOM "cf rrt=en"), the RTC Interface will not know about this change and will continue to try to operate according to the earlier setting. In this case, the RTC Interface may try to update its displays when the emulator is set to deny monitor access to the registers and memory.
- If you have a command in your command file that writes a value to memory (such as, TERMCOM "00000..00fff=0"), the Memory window will be updated immediately to show the new value, assuming you have chosen RealTime→Monitor Intrusion→Allowed.

Do not use the following Terminal Interface commands with the RTC TERMCOM command:

- stty, po, xp: These commands will change the operation of the communications channel, and are likely to hang the RTC Interface.
- echo, mac: These commands may confuse the communications protocols in use in the channel.
- wait: The pod will enter a wait state, blocking access by the RTC Interface.
- init, pv: These will reset the emulator and end your session.
- t: This will confuse the functions of trace status polling and unload.

Refer to your "Terminal Interface User's Guide" for more information about Terminal Interface commands.

Command File Command

```
TERMCOM "ti-command"
```

WAIT

Inserts wait delays during command file execution.

Command File Command

`WAI(T) MON(ITOR)`
Waits until MONITOR status.

`WAI(T) RUN`
Waits until RUN status.

`WAI(T) UNK(NOWN)`
Waits until UNKNOWN status.

`WAI(T) SLO(W)`
Waits until SLOW CLOCK status.

`WAI(T) TGT(RESET)`
Waits until TARGET RESET status.

`WAI(T) SLE(EP)`
Waits until SLEEP status.

`WAI(T) GRA(NT)`
Waits until BUS GRANT status

`WAI(T) NOB(US)`
Waits until NOBUS status.

`WAI(T) TCO(M)`
Waits until the trace is complete.

`WAI(T) THA(LT)`
Wait until the trace is halted.

`WAI(T) TIM(E) seconds`
Waits for a number of seconds.

12

Error Messages

Error Messages

This chapter helps you find details about the following error messages:

- Bad RS-232 port name
- Bad RS-422 card I/O address
- Could not open initialization file
- Could not write Memory
- Error occurred while processing Object file
- General RS-232 communications error
- General RS-422 communications error
- HP 64700 locked by another user
- HP 64700 not responding
- Incorrect DLL version
- Incorrect LAN Address (HP-ARPA, Windows for Workgroups)
- Incorrect LAN Address (Novell)
- Incorrect LAN Address (WINSOCK)
- Internal error in communications driver
- Internal error in Windows
- Interrupt execution (during run to caller)
- Interrupt execution (during step)
- Interrupt execution (during step over)
- Invalid transport name
- LAN buffer pool exhausted
- LAN communications error
- LAN MAXSENDSIZE is too small

- LAN Socket error
- Object file format ERROR
- Out of DOS Memory for LAN buffer
- Out of Windows timer resources
- PC is out of RAM memory
- Timed out during communications

Refer to the end of Chapter 4 for further discussions of specific error conditions.

Bad RS-232 port name

RS-232 port names must be of the form "COM<number>" where <number> is a decimal number from 1 to the number of communications ports within your PC.

Bad RS-422 card I/O address

The RS-422 card's I/O address must be a hexadecimal number from 100H through 3F8H whose last digit is 0 or 8 (example 100, 108, 110). Select an I/O address that does not conflict with the other cards in your PC.

Could not open initialization file

The initialization file was not found in the same directory where the executable file was found.

For example, if the application file is b3621.EXE, the initialization file b3621.INI is expected to be found in the same directory.

To fix this problem, you may be able to find the initialization file and move it to the same directory as the executable file, or you can create a new initialization file from the default initialization file. For example:

COPY b3621DEF.INI Bxxxx.INI

Note that the above command is the DOS COPY command. Do not use the ksh 'cp b3621DEF.INI Bxxxx.INI' command. Use only the DOS 'COPY b3621DEF.INI b3621.INI' command.

If you cannot find the default initialization file either, you can re-install the debugger software.

For correct operation, make certain the b3621.INI file has both read and write permission.

Could not write Memory

You may see this error message when trying to load a file or perform any other task that requires use of the monitor. The emulation monitor is used to load files, which requires writing to memory. If you have chosen RealTime→Monitor Intrusion→Disallowed the monitor will not be usable, and Execution→Reset may prevent use of the monitor in some emulators.

Choose RealTime→Monitor Intrusion→Allowed, and Execution→Break to ensure that the emulation monitor is running. The Status window should show Emulator: RUNNING IN MONITOR.

With this setup, the emulator should be able to write to Memory.

If you are still unable to load a file, select "Symbols Only" in the Load Object File dialog box and try to load the file. If Symbols Only will not load, the problem is in your symbols.

Choose "Data Only" in the Load Object File dialog box and try to load the file. If the symbols loaded, but the data fails to load, the problem is in your program code.

Call your local HP representative.

Error occurred while processing Object file

The following is a list of typical reasons why an error might occur while processing an object file. There are many other possible reasons.

- Bad record in the object file.
- File is in wrong format.
- File does not follow OMF Specifications correctly.
- No memory mapped.
- Attempt to write to guarded memory.
- Emulator restricted to real-time runs. Enter the command, "RealTime→Monitor Intrusion→Allowed".
- Emulator not executing the monitor. Enter the command, "Execution→Break".

Another message often occurs along with this message. View the help information for the other message, if available.

Call your local HP representative.

General RS-232 communications error

In general, these messages indicate that the RS-232 communication has intermittent errors. Sometimes you will get this message if you power on the emulator, or when you try to connect to the emulator. In that case, simply retry the connection (by double-clicking on the RS232C driver line in the selection box); if you connect with no problems the second time, you can ignore the original message.

If you get this message other than during connection, you can try to fix the problem by:

- Reducing the length of the RS-232 cable between the PC and the HP 64700.
- Reducing the number of tasks running under Windows.
- Reducing the baud rate (the default is 19200).

For further information, refer to the paragraph titled, "If you have RS-232 connection problems" in the Communications Help screen, or in Chapter 15, "Installing the Debugger" in the Real-Time C Debugger User's Guide.

General RS-422 communications error

In general, these messages indicate that the RS-422 communication has intermittent errors. Sometimes you will get this message if you power on the emulator, or when you try to connect to the emulator. In that case, simply retry the connection (by double-clicking on the HP-RS422 driver line in the selection box); if you connect with no problems the second time, you can ignore the original message.

If you get this message other than during connection, you can try to fix the problem by:

- Reducing the number of tasks running under Windows.
- Reducing the baud rate (the default is 230400).

HP 64700 locked by another user

Because it is possible to destroy another user's measurement by choosing the Unlock button in the error dialog box, check with the other user before unlocking the HP 64700.

Note that if the other user is actually using an interface to the HP 64700, an Unlock request will fail.

HP 64700 not responding

The HP 64700 has not responded within the timeout period. There are several possible causes of this error. For example, a character could have dropped during RS-232 communications, or some network problem could have disrupted communications.

Usually, you must cycle power to the HP 64700 to fix this problem.

See also: The description for the error message titled, "Timed out during communications."

Incorrect DLL version

The version of the dynamic link libraries (.DLLs) used by the Real-Time C Debugger does not match the version of the main program (.EXE).

If you have two versions of the debugger on your system, you may see this message when you try to execute both of them at the same time, or when you execute one version and then the other without restarting Windows. Once DLLs have been loaded into Windows memory, they stay there until you exit Windows. Therefore, exit windows, restart windows, and try again.

This message will also appear if you have somehow loaded a version of the DLLs that is different from the version of the executable. In this case, you must reload your software.

Incorrect LAN Address (HP-ARPA, Windows for Workgroups)

A LAN address can be one of two types: an IP address, or a host name.

An IP address consists of four digits separated by dots. Example:

```
15.6.28.0
```

A hostname is a name that is related (mapped) to an IP address by a database. For example, the file \LANMAN.DOS\ETC\HOSTS (HP-ARPA) or \WINDOWS\HOSTS (Windows for Workgroups) may contain entries of the form:

```
system1 15.6.28.0
```

Note

The directory of the "hosts" file may be different on your system.

If "HP Probe" or "DNR" (Domain Name Resolution) is available on your PC, those are consulted first for a mapping between the hostname and the IP address. If the hostname is not found by that method, or if those services are unavailable, the local "hosts" file is consulted for the mapping.

Note that if "Probe" is available on your system but unable to resolve the address, there will be a delay of about 15-seconds while Probe is attempting to find the name on the network.

Incorrect LAN Address (Novell)

A LAN address can be one of two types: an IP address, or a host name.

An IP address consists of four digits separated by dots. Example:

```
15.6.28.0
```

A hostname is a name that is related (mapped) to an IP address by a database. For example, the file \NET\TCP\HOSTS may contain entries of the form:

```
system1 15.6.28.0
```

Note

The directory of the "hosts" file may be different on your system. Also, all files defined by the PATH TCP_CFG setting under "Protocol TCPIP" in the NET.CFG files are searched.

Incorrect LAN Address (WINSOCK)

A LAN address can be one of two types: an IP address, or a host name.

An IP address consists of four digits separated by dots. Example:

```
15.6.28.0
```

A hostname is a name that is related (mapped) to an IP address by a database. For example, the hosts file may contain entries of the form:

```
system1 15.6.28.0
```

Note

Because WINSOCK is a standard interface to many LAN software vendors, you need to read your LAN vendor's documentation before specifying the LAN address.

Internal error in communications driver

These types of errors typically occur because other applications have used up a limited amount of some kind of global resource (such as memory or sockets).

You usually have to reboot the PC to free the global resources used by the communications driver.

Internal error in Windows

These types of errors typically occur because other applications have used up a limited supply of some kind of global resource (such as memory, sockets, tasks, or handles).

You usually have to reboot the PC to free the global resources used by Windows.

Interrupt execution (during run to caller)

The Return dialog box appears when running to the caller of a function and the caller is not found within the number of milliseconds specified by StepTimerLen in the .INI file of the debugger application.

You can cancel the run to caller command by choosing the STOP button, which causes program execution to stop, the breakpoint to be deleted, and the processor to transfer to the RUNNING IN USER PROGRAM status.

Interrupt execution (during step)

The Step dialog box appears when stepping a source line or assembly instruction and the source line or instruction does not execute within the number of milliseconds specified by StepTimerLen in the .INI file of the debugger application.

You can cancel the step command by choosing the STOP button, which causes program execution to stop, the breakpoint to be deleted, and the processor to transfer to the RUNNING IN USER PROGRAM status.

Interrupt execution (during step over)

The Step dialog box appears when stepping over a function or subroutine and the function or subroutine does not execute within the number of milliseconds specified by StepTimerLen in the .INI file of the debugger application.

You can cancel the step-over command by choosing the STOP button, which causes program execution to stop, the breakpoint to be deleted, and the processor to transfer to the RUNNING IN USER PROGRAM status.

Invalid transport name

The transport name chosen does not match any of the possible transport names (RS232C, HP-ARPA, Novell-WP, WINSOCK1.1, W4WG-TCP, or HP-RS422).

The transport name can be specified either on the command line with the -t option or in the .INI file:

[Port]
Transport=<transport name>

Choosing an appropriate transport in the dialog box that follows this error message will correct the entry in the .INI file, but if the error is in the command line option, you must modify the command line (by using the "Properties..." command in the Program Manager).

LAN buffer pool exhausted

The LAN buffer pool is used as a temporary buffer between the time the debugger sends data and the time the LAN actually sends it. When this pool is exhausted, the debugger cannot send any data across the LAN.

The size of the sockets buffer pool is configured in the network installation procedure. The size and number of LAN buffer pools can be changed by editing your network configuration file.

LAN communications error

This message may appear after any kind of LAN error.

Refer to the documentation for your LAN software for descriptions of the types of problems that can cause LAN errors.

LAN MAXSENDSIZE is too small

This message indicates you have configured your LAN with a value or MAXSENDSIZE that is less than 100 bytes. Note that the default is 1024 bytes.

The Real-Time C Debugger requires at least 100 bytes for this parameter.

To fix this, change the following entry in your PROTOCOL.INI file and reboot your PC:

[SOCKETS]
MAXSENDSIZE

LAN socket error

A TCP-level error has occurred on the network. See your network administrator.

Object file format ERROR

This message is typically caused by one of two conditions:

- Bad format file. Perhaps there is a bad record within the file. If you have a file format verifier, submit your file to it to determine whether or not all records are in the correct format.
- Unknown construct. Perhaps the construct of your file is unfamiliar to the reader.

To respond to this error message, verify the file format, and ensure that the reader can understand the file format in use.

If these steps do not solve the problem, call your local HP representative.

Out of DOS Memory for LAN buffer

This means that there is not enough memory in the lower 1 Mbyte of address space (that is, conventional memory) for the LAN driver to allocate a buffer to communicate with the LAN TSR.

When you are in windows, and execute the DOS command "mem", you cannot see the memory that is in the lower 1 Mbyte that is used by the windows program. If you have the Microsoft program "heapwalker", you can use it to see what programs have allocated space in the address range 0 through FFFFF.

To fix this, you can:

- Reduce the number of TSRs running on your PC (before Windows starts) that use conventional memory.
- Reconfigure your network to have fewer sockets or modules loaded, or to be configured for fewer total connections.
- Use a different memory manager to reduce your network memory usage, such as QEMM.

Out of Windows timer resources

The debugger is not able to acquire the timer resources it needs.

There are a limited number of timer resources in Windows. You may be able to free timer resources by closing other applications.

PC is out of RAM memory

The debugger is not able to acquire the memory it needs because other applications are using it, or because of fragmented memory.

You may be able to free memory by closing other applications, or you might have to reboot the PC to cause memory to be unfragmented.

Timed out during communications

The HP 64700 has not responded within the timeout period. There are various causes for this error. For example, a character could have been dropped during RS-232 communications or some network problem could have disrupted communications.

The timeout period for reading and writing to the HP 64700 is defined by TimeoutSeconds in either the [RS232C], [HP-ARPA], [Novell-WP], or [HP-RS422] section of the b3621.INI file. For example, if you are using the RS-232C transport:

```
[RS232C]
TimeoutSeconds=<seconds>
```

The number of seconds can be between 1 and 32767. The default is 20 seconds.

If you are using RS-232C or RS-422 transport ...

The TimeoutSeconds value is also used for connecting to the HP 64700 (as well as for reading and writing).

If you are using HP-ARPA or Novell-WP transport ...

If there are several gateways or bridges between the PC and the emulator, larger values of TimeoutSeconds may be reasonable.

The timeout period for connecting to the HP 64700 is defined in the PROTOCOL.INI file.

```
[TCPIP_XFR]
TCPCONNTIMEOUT=<seconds>
```

The default connection timeout is 30 seconds.

The remainder of this discussion shows you how to overcome the problem of "connection timed out" during large memory fill operations.

The RTC interface sends the memory fill operation to the emulator as a single command. While the command is executing in the emulator, the emulator cannot respond to inquiries from the interface about its status. If the memory fill takes long enough, the connection will time out.

Emulators for some microprocessors take up to one minute per megabyte to perform a memory fill operation. Timeout default values for RTC interfaces shipped from HP are typically 45 seconds.

First Workaround. Modify the TimeoutSeconds field (discussed above) to increase the TimeoutSeconds value. Then exit the interface and restart it (to ensure that the new value of TimeoutSeconds is read). You may experiment with several values of TimeoutSeconds to find the value that allows you to do a memory fill. The problem with this workaround is that all timeouts will take this new longer time, and you may find this annoying when you are not doing memory fill operations.

Second Workaround. Create a command file that contains TERMCOM commands to write to small portions of the overall memory to be filled. For example, suppose the following Memory window command causes the emulator to time out, "Memory→Utilities→Fill→0 to ffff".

You might make a command file named memfill.cmd, and place the following commands in it:

```
TERMCOM "m 00000..00fff=0"
TERMCOM "m 01000..01fff=0"
TERMCOM "m 02000..02fff=0"
TERMCOM "m 03000..03fff=0"
TERMCOM "m 04000..04fff=0"
TERMCOM "m 05000..05fff=0"
TERMCOM "m 06000..06fff=0"
TERMCOM "m 07000..07fff=0"
TERMCOM "m 08000..08fff=0"
TERMCOM "m 09000..09fff=0"
TERMCOM "m 0a000..0afff=0"
TERMCOM "m 0b000..0bfff=0"
TERMCOM "m 0c000..0cfff=0"
TERMCOM "m 0d000..0dfff=0"
TERMCOM "m 0e000..0efff=0"
TERMCOM "m 0f000..0ffff=0"
```

When you choose File→Run Cmd File→... and select your memfill.cmd file, it will not exceed the timeout value. This is because the emulator will be able to respond to inquiries from the interface between execution of each of the TERMCOM commands in your command file.

Part 4

Concept Guide

Topics that explain concepts and apply them to advanced tasks.

13

Concepts

Concepts

This chapter describes the following topics.

- Debugger Windows
- Compiler/Assembler Specifications
- Monitor Programs
- Trace Signals and Predefined Status Values

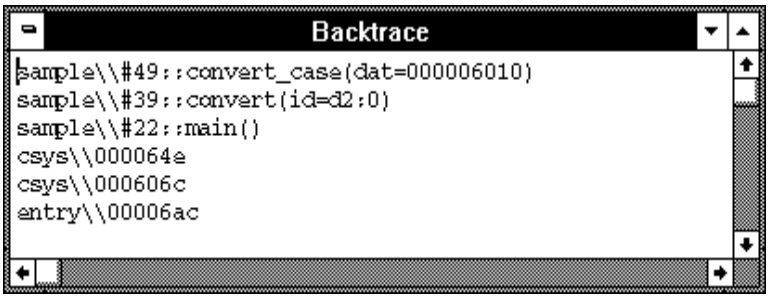
Debugger Windows

This section describes the following debugger windows:

- BackTrace
- Button
- Device Regs
- Expression
- I/O
- Memory
- Register
- Source
- Status
- Symbol
- Trace
- WatchPoint

The BackTrace Window

The BackTrace window displays the function associated with the current program counter value and this function's caller functions in backward order. Applicable addresses are prefixed with module\#linenum information. The current arguments of these functions are also displayed.



The BackTrace window is updated when program execution stops at an occurrence of breakpoint, break, or Step command.

The BackTrace window lets you copy text strings, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

By clicking the right mouse button in the BackTrace window, you can access the Source at Stack Level popup menu command. Cursor-select a function in the BackTrace window and choose this command to display (in the Source window) the code that called the function.

See Also

"BackTrace Window Pop-Up Commands" in the "Window Pop-Up Commands" chapter.

The Button Window

The Button window contains user-defined buttons that, when chosen, execute debugger commands or command files.

The Button window's *control menu* provides the Edit... (ALT, -, E) command which lets you add and delete buttons from the window.

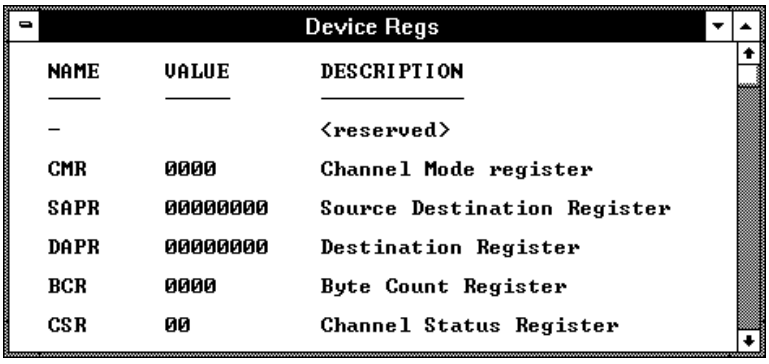
See Also

"Using Command Files" in the "Using the Debugger Interface" chapter.

"Button Window Commands" in the "Window Control Menu Commands" chapter.

Device Regs Window

The Device Regs window shows all of the internal registers that are used to control various devices. Each register is represented by a row which holds a mnemonic name, a current value, and a description of the register contents.



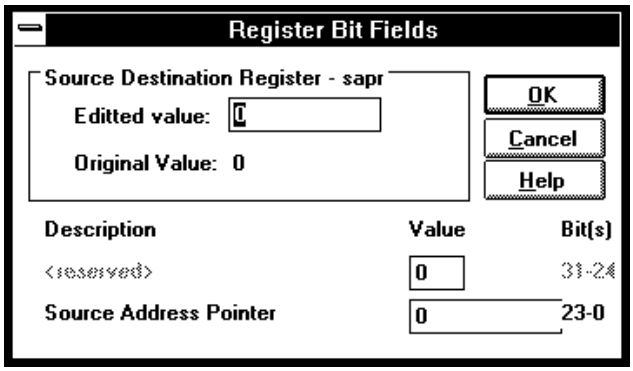
The registers may be edited by either single clicking or double-clicking on the value. A single click puts you in a mode where the left or right arrow keys may be used for placement of the cursor. Double-clicking puts you in one of two modes; either a Register Bit Fields dialog pops up or the value is highlighted. When the value is highlighted, the backspace key will erase the value and a completely new value may be entered. This mode is applicable to registers where the value is considered a single number and is not divided by any bit-fields.

See Also

"Device Regs Window Commands" in the "Window Control Menu Commands" chapter.

Device Register Dialogs

When a register has bit-fields, a dialog will pop-up and the register value may be edited by changing the whole value or by editing individual bit-fields.

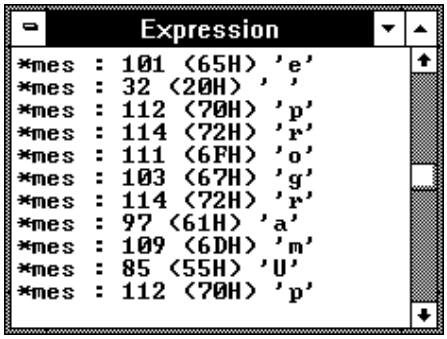


When editing in the dialog box, a carriage-return is the same as choosing the OK button. To end an edit of a field within the dialog box without quitting, use the Tab key.

Edited Value	Shows the register value that corresponds to the selections made below. You can also change the register's value by modifying the value in this text box.
Original Value	Shows the value of the register when the dialog box was opened. If the register could not be read, 'XXXXXXXX' is displayed.
OK	Modifies the register as specified, and closes the dialog box.
Cancel	Closes the dialog box without modifying the register.

The Expression Window

The Expression window displays the results of the EVALUATE commands in command files or break macros.



When a variable name is specified with the EVALUATE command, the Expression window displays the evaluation of the variable. When a quoted string of ASCII characters is specified with the EVALUATE command, the Expression window displays the string.

The Expression window's *control menu* provides the Evaluate... (ALT, -, E) command which lets you evaluate expressions and see the results in the window.

See Also

"Expression Window Commands" in the "Window Control Menu Commands" chapter.

The I/O Window

The I/O window displays the contents of the I/O locations.

You can modify the contents of I/O locations by double-clicking on the value, using the keyboard to type in the new value, and pressing the Enter key.

The I/O window contents are updated periodically when the processor is running the user program.

If a location is in target system memory, a temporary break from the user program into the monitor program must occur in order for the debugger to update or modify that location's contents. If it's important that the user program execute without these types of interruptions, you should disallow monitor intrusion. Even when monitor intrusion is allowed, you can stop temporary breaks during the window update by turning polling OFF.

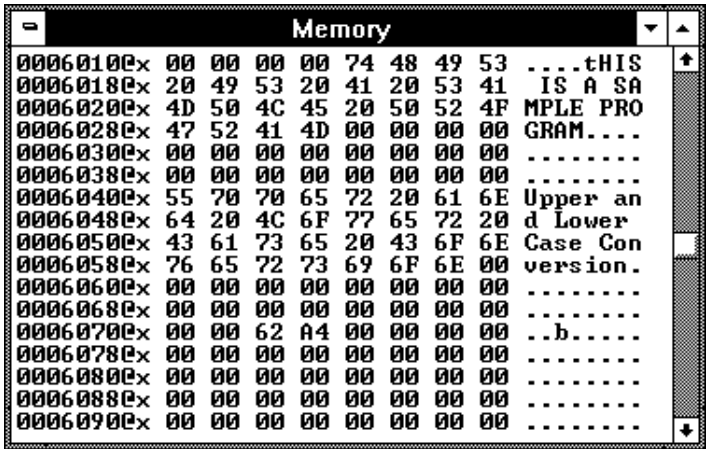
See Also

"Displaying and Editing I/O Locations" in the "Debugging Programs" chapter.

"I/O Window Commands" in the "Window Control Menu Commands" chapter.

The Memory Window

The Memory window displays memory contents.



The Memory window has *control menu* commands that let you change the format of the memory display and the size of the locations displayed or modified. When the absolute (single-column) format is chosen, symbols corresponding to addresses are displayed. When data is displayed in byte format, ASCII characters for the byte values are also displayed.

When Memory window polling is turned ON, you can modify the addresses displayed or contents of memory locations by double-clicking on the address or value, using the keyboard to type in the new address or value, and pressing the Enter key.

The Memory window contents are updated periodically when the processor is running the user program.

If a location is in target system memory, a temporary break from the user program into the monitor program must occur in order for the debugger to update or modify that location's contents. If it's important that the user program execute without these types of interruptions, you should disallow monitor intrusion. Even when monitor intrusion is allowed, you can stop temporary breaks during the window update by turning polling OFF.

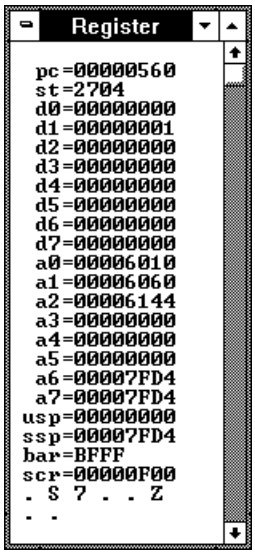
See Also

"Displaying and Editing Memory" in the "Debugging Programs" chapter.

"Memory Window Commands" in the "Window Control Menu Commands" chapter.

The Register Window

The Register window displays contents of registers.



You can modify register contents by double-clicking on the register value, using the keyboard to type in the new value, and pressing the Enter key. When you double-click on the Status Register (st), System Control Register (scr), or Base Address Register (bar) values, a dialog box opens and allows you to set or clear individual bits.

The Register window contents are updated periodically when the processor is running the user program and monitor intrusion is allowed.

A temporary break from the user program into the monitor program must occur in order for the debugger to update or modify register contents. If it's important that the user program execute without these types of interruptions, you should disallow monitor intrusion.

See Also

"Displaying and Editing Registers" in the "Debugging Programs" chapter.

"Register Window Commands" in the "Window Control Menu Commands" chapter.

The Source Window

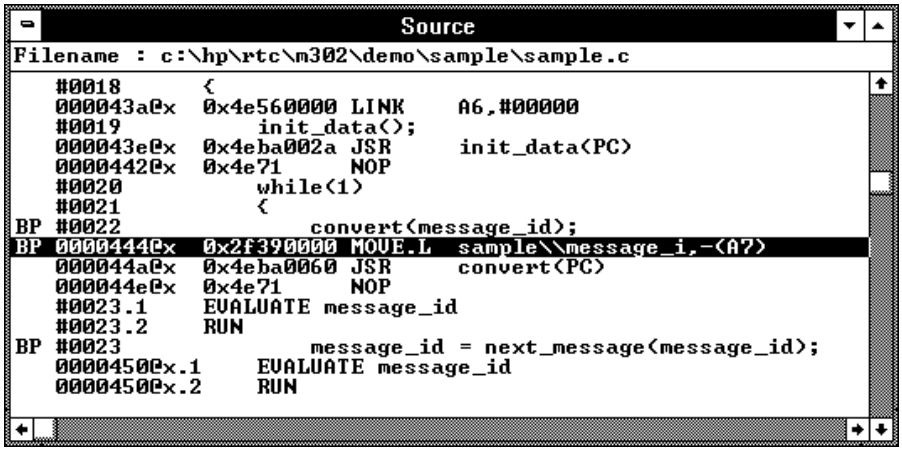
The Source window displays source files, optionally with disassembled instructions intermixed.

The Source window contains a cursor whose position is used when setting or deleting breakpoints or break macros or when running the program up to a certain line.

The Source window lets you copy strings, usually variable or function names to be used in commands, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

The Source window also provides commands in the *control menu* that let you select whether disassembled instruction mnemonics should appear intermixed with the C source code.

By clicking the right mouse button in the Source window, you can also access pop-up menu commands.



Filename	The name of the displayed source file appears at the top of the window.
Source Lines	C source code is displayed when available. Source lines are preceded by the corresponding line numbers. When programs are written in assembly language or when no C source code is available, disassembled instruction mnemonics are displayed. The interface will only support display in either trace or source windows of source lines numbered less than 32,000.
Disassembled Instructions	In the Mnemonic Display mode, disassembled instruction mnemonics are intermixed with the source lines. Disassembled lines contain address, data, and mnemonic information. When symbolic information is available for the address, the corresponding symbol line precedes the disassembled instruction, displayed in the module_name\\symbol_name format.
Current PC	The line associated with the current program counter is highlighted.
Scroll Bars	For C source files, the display scrolls within the source files. For assembly language programs or programs for which no source code is available, the display scrolls for all the memory space.
"BP" Marker	The breakpoint marker, "BP", appears at the beginning of the breakpoint lines or break macro lines.
Execution Coverage	The accessed (executed) lines are highlighted when program execution coverage is enabled. This is not available when using the HP 64170 emulation memory board.

Break Macro Lines	Decimal points following line numbers or addresses indicate break macro lines.

Note

When programs are stored in target system memory and the emulator is running in real-time, source code cannot be displayed.

See Also

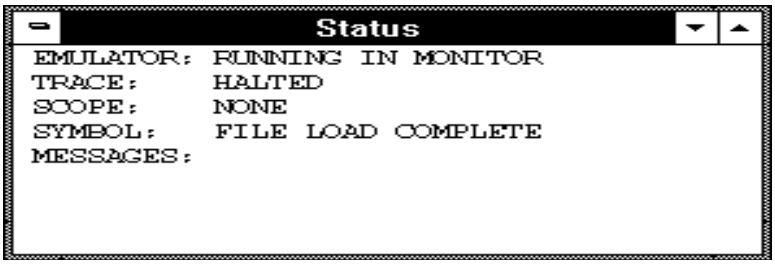
"Loading and Displaying Programs",
"Stepping, Running, and Stopping the Program",
"Using Breakpoints and Break Macros", and
Making Coverage Measurements" in the "Debugging Programs" chapter.

"Source Window Commands" in the "Window Control Menu Commands" chapter.
"Source Window Pop-Up Commands" in the "Window Pop-Up Commands" chapter.
"To set colors in the Source window" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

The Status Window

The Status window shows:

- Emulator status.
- Trace status.
- Scope of the current program counter value.
- Progress of symbols being loaded from a file.
- Last five asynchronous messages from the emulator.



Emulation Processor Status Messages

`EMULATION RESET`
The emulation processor is being held in the reset state by the emulator.

`RUNNING IN MONITOR`
The emulation processor is executing the monitor program.

`RUNNING IN USER PROGRAM`
The emulation processor is executing the user program.

`RUNNING REALTIME IN USER PROGRAM`
The emulation processor is executing the user program in the real-time mode where:

- Any command that would temporarily interrupt user program execution is disabled.

- Any on-screen information that would be periodically updated by temporarily interrupting user program execution (target system memory or register contents, for example) is disabled.

`WAITING FOR TARGET RESET`
The emulation processor is waiting for a RESET signal from the target system. User program execution starts on reception of the RESET signal.

`SLOW CLOCK`
No proper clock pulse is supplied from the external clock.

`EMULATION RESET BY TARGET`
The emulation processor is being held in a reset state by a RESET signal from the target system.

`BUS GRANT TO TARGET SYSTEM DEVICE`
The bus is granted to some device in the target system.

`NO BUS CYCLE`
The bus cycle is too slow or no bus cycle is provided.

`HALTED`
The emulation processor has halted.

`UNKNOWN STATE`
The emulation processor is in an unknown state.

Other Emulator Status Messages

The Status window may also contain status messages other than the emulation processor status messages described above:

`BREAK POINT HIT AT module_name#line_number`
The breakpoint specified in the source code line was hit and program execution stopped at "line_number" in "module".

`BREAKPOINT HIT AT address`
The breakpoint specified in the assembled line was hit and program execution stopped at "address".

`UNDEFINED BREAKPOINT at address`
The breakpoint instruction occurred at "address", but it was not inserted by a breakpoint set command.

`WRITE TO ROM BREAK`
Program execution has stopped due to a write to location mapped as ROM. These types of breaks must be enabled in the emulator configuration.

`ACCESS TO GUARD BREAK`
Program execution has stopped due to a write to a location mapped as guarded memory.

`TRACE TRIGGER BREAK`
The analyzer trigger caused program execution to break into the monitor (as specified by selecting the Break On Trigger option in the trace setting dialog box).

Trace Status Messages

`TRACE RUNNING`
The trace has been started and trace memory has yet to be filled; this could be because the trigger condition has not occurred or, if the trigger condition has occurred, there have not been enough states matching the store condition to fill trace memory. Contents of the trace buffer cannot be displayed during the TRACE RUNNING status; you must halt the trace before you can display the contents of the trace buffer.

`TRACE HALTED`
The trace was halted before the trace buffer was filled. The status indicates that the trace was halted immediately after the emulator powerup, or that the trace was force-terminated by the user. In the TRACE HALTED status, the analyzer displays the contents of the trace buffer before the halt in the Trace window.

`TRACE COMPLETE`
The trace completed because the trace buffer is full. The results are displayed in the Trace window.

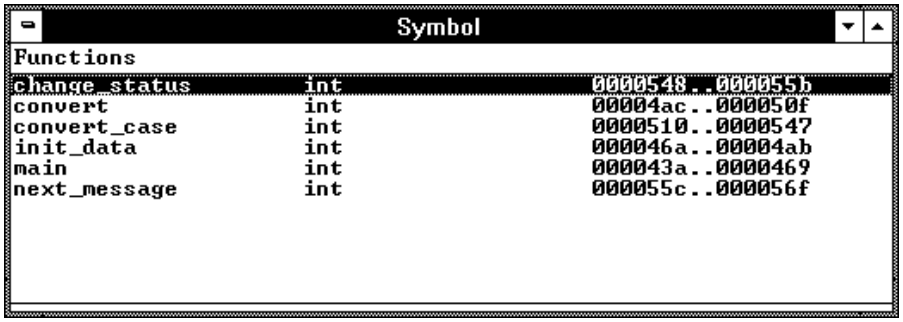
The Symbol Window

The Symbol window displays information on the following types of symbols:

- Modules
- Functions
- Global symbols
- Local symbols
- Global Assembler symbols
- Local Assembler symbols
- User-defined symbols

The Symbol window has *control menu* commands that let you display various types of symbols, add or delete user-defined symbols, copy Symbol window information, or search for symbols that contain a particular string.

The Symbol window lets you copy symbols to the clipboard by clicking the left mouse button. The symbol information can then be pasted from the clipboard in other commands.



Symbols are displayed with "type" and "address" values where appropriate.

See Also

"Displaying Symbol Information" in the "Debugging Programs" chapter.

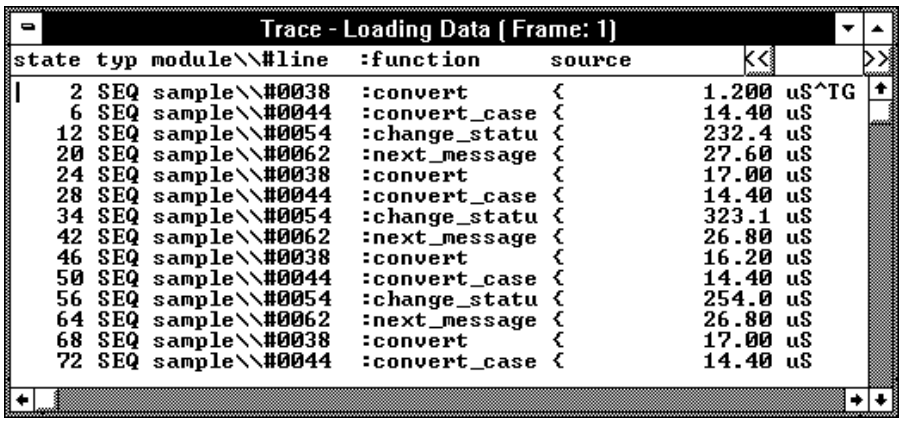
Symbol Window Commands" in the "Window Control Menu Commands" chapter.

The Trace Window

The Trace window displays trace results and shows source code lines that correspond to the execution captured by the analyzer. Optionally, bus cycle states can be displayed along with the source code lines.

The Trace window has *control menu* commands that let you display bus cycles, specify whether count information should be absolute or relative, or copy information from the window.

The Trace window opens automatically when a trace is complete.



For each line in the Trace window, the trace buffer state number, the type of state, the module name and source file line number, the function name, the source line, and the time count information are displayed.

The << and >> buttons let you move between the multiple frames of trace data that are available with newer analyzers for the HP 64700.

The type of state can be a sequence level branch (SEQ), a state that satisfies the prestore condition (PRE), or a normal state that matches the store conditions (in which case the type field is empty).

Bus cycle states show the address and data values that have been captured as well as the disassembled instruction or status mnemonics.

On startup, the system defaults to the source only display mode, where only source code lines are displayed. The source/bus cycle mixed display mode

can be selected by using the Trace window control menu's Display→Mixed Mode (ALT, -, D, M) command. In the source/bus cycle mixed display mode, each source code line is immediately followed by the corresponding bus cycles.

The trace buffer stores bus cycles only. The system displays source lines in the Trace window based on execution bus cycles.

See Also

"Tracing Program Execution" and
"Setting Up Custom Trace Specifications" in the "Debugging Programs" chapter.

"Trace Window Commands" in the "Window Control Menu Commands" chapter.

The WatchPoint Window

The WatchPoint window displays the contents of variables that have been registered with the Variable→Edit... (ALT, V, E) command or with the Edit... (ALT, -, E) command in the WatchPoint window's control menu.

The contents of dynamic variables are displayed only when the current program counter is in the function in which the variable is declared.

You can modify the contents of variables by double-clicking on the value, using the keyboard to type in the new value, and pressing the Enter key.

The WatchPoint window lets you copy text strings, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

See Also

"Displaying and Editing Variables" in the "Debugging Programs" chapter.

"WatchPoint Window Commands" in the "Window Control Menu Commands" chapter.

Compiler/Assembler Specifications

This section describes:

- IEEE-695 Object Files
- Compiling Programs with MCC68K
- Compiling Programs with AxLS

IEEE-695 Object Files

This section addresses the IEEE-695 object files compiled or assembled with the following compilers and assemblers:

- Microtec MCC68K Compiler
- Microtec ASM68K Assembler
- HP AxLS Compiler
- HP AxLS Assembler

Assembly Language Source File Display

The IEEE-695 object files do not contain assembly language source file information. Instead, memory contents are disassembled.

Mnemonic Display

An assembly language instruction preceding or following a function entry point may have multiple corresponding source code lines. For this type of instruction, the Source window in the Mnemonic Display mode shows multiple corresponding disassembled lines having the same address.

Single-Stepping Loop Control Statements

The system may fail in single-stepping such loop control statements as "while", "for", or "do while" statement.

Pragma Statement and Debugger Display

When a "pragma" statement is used to describe an assembly language instruction in C source files, the source information is generated as follows in the IEEE-695 object files:

- A pragma instruction has a single line number.
- The address for the pragma instruction indicates the address for the first line of the instruction.
- The line number for the pragma instruction indicates the line number for the last line of the instruction.

This imposes the following display restriction on the Real-Time C Debugger:

The Source window in the Mnemonic Display mode shows lines in a pragma instruction all at one time as listed below.

```
#0010      #pragma asm
#0011           nop
#0012           nop
#0013      #pragma endasm
0001000    00         NOP
0001001    00         NOP
```

During single-stepping, the last line of the pragma instruction is highlighted while the program counter indicates the first line.

```
#0010      #pragma asm
#0011           nop
#0012           nop
#0013      #pragma endasm
```

Program counter indicating line 11

Highlighted line 12

Only the last line of the pragma instruction is displayed in the trace results.

Compiling Programs with MCC68K

1 Compile the source files with the mcc68k command.

2 Assemble the source files with the asm68k command.

3 Link the object files with the lnk68k command.

Required Compiler/Assembler/Linker

Compiler	Microtec MCC68K Compiler
Assembler	Microtec ASM68K Assembler
Linker	Microtec LNK68K Linker

Compiling

For compiling, use the mcc68k command in your Microtec C Compiler with the following option switches:

-g	Outputs debugging information.
-Gf	Generates fully-qualified path names for input files.
-nOg	Disables global flow optimization.
-nOR	Disables register variables.
-Kf	Creates frame pointers for functions.

Note The -nOg and -nOR options allow the debugger to display arguments during backtracing.

Note The -Kf option allows the debugger to trace function flow.

Assembling

For assembling, use the asm68k command in your Microtec Assembler with the following option switch:

-fd	Creates local symbols.

Linking

For linking, use the lnk68k command in your Microtec Linker. Specify the IEEE-695 file format for the load module.

Example

To compile and link sample.c user program into a load module, execute the following command, where sample.k is the linker command file:

```
A> mcc68k -g -Gf -Kf -nOg -nOR -l -esample.k -osample.x
sample.c -Wl,-m > sample.lst
```

Compiling Programs with AxLS

1. Compile the source files with the cc68000 command.
2. Assemble the source files with the as68k command.
3. Link the object files with the ld68k command.

Required Compiler/Assembler/Linker

Compiler	HP AxLS CC68000 Compiler
Assembler	HP AxLS AS68K Assembler
Linker	HP AxLS LD68K Linker

Compiling

For compiling, use the cc68000 command in your HP AxLS C Compiler with the following option switches:

-Wc,-F Disables register variables.

Note

The -Wc,-F option allows the debugger to display arguments during backtracing.

Assembling

For assembling, use the as68k command in your HP AxLS Assembler without any option switch.

Linking

For linking, use the ld68k command in your HP AxLS Linker. Specify the IEEE-695 file format for the load module.

Note

The Real-Time C Debugger does not support simulated I/O locations. You can use the -N compiler option to use a linker command file that does not include the simulated I/O library.

Example

To compile and link sample.c user program into a load module, execute the following command, where sample.k is the linker command file:

```
cc68000 -N -Wc,-F -Lix -k sample.k -o sample.x sample.c
```

Monitor Programs

This section describes:

- Monitor Program Options
- Assembling and Linking the Foreground Monitor with MCC68K
- Assembling and Linking the Foreground Monitor with AxLS
- Setting Up the Trace Vector
- Notes on Foreground Monitors

The foreground monitor source file is included with the debugger software and can be found in the C:\HP\RTC\M302\FGMON directory (if C:\HP\RTC\M302 was the installation path chosen when installing the debugger software).

Monitor Program Options

The emulation monitor program is a program that the emulation microprocessor executes as directed by the HP 64700 system controller. The emulation monitor program gives the system controller access to the target system.

For example, when you modify target system memory, the system controller writes a command code to a communications area and switches, or breaks, emulation processor execution into the monitor program. The monitor program reads the command code (and any associated parameters) from the communications area and executes the appropriate machine instructions to modify the target system locations. After the monitor has performed its task, emulation processor execution returns to what it was doing before the break.

The emulation monitor program can execute out of a separate, internal memory system known as background memory. A monitor program executing out of background memory is known as a background monitor program.

The emulation monitor program can also execute out of the same memory system as user programs. This memory system is known as foreground memory and consists of emulation memory and target system memory. A monitor program executing out of foreground memory is known as a foreground monitor program. Foreground monitor programs must exist in emulation memory.

The emulator firmware includes the background monitor. You can also load and use a foreground monitor program if needed.

Background Monitor

The default emulator configuration selects the background monitor.

Interrupts from the target system are disabled during background monitor execution. If your programs have strict real-time requirements for servicing target system interrupts, you must use a foreground monitor program.

Foreground Monitor

A foreground monitor source file is provided with the emulator. It can be assembled, linked, and loaded into the debugger.

A foreground monitor has the following advantages and disadvantages:

Advantages

- The foreground monitor executes as a part of the user program, and target system interrupts can be enabled during monitor program execution for applications that have strict real-time processing requirements.
- The foreground monitor can be customized.

Disadvantages

- The foreground monitor occupies 2 Kbytes of the user memory space.
- The foreground monitor must be assembled and linked prior to use.

Addressing for Foreground Monitor

The foreground monitor is loaded into emulation memory just like a user program. Assemble and link the foreground monitor at an address space not used by the user program.

The starting address can be specified on a 2 Kbyte boundary starting from 800H. If you're using the HP 64170 emulation memory board with 1 Mbyte memory modules, the starting address must be specified on an 8 Kbyte boundary.

To specify the foreground monitor starting address, you must modify the ORG statement that follows the first comment in the source program as shown below.

```
; ORG XXXXXXH   ; START MONITOR ON 2K BOUNDARY OTHER THAN ZERO
```

Specify the foreground monitor address by removing the semicolon (;) before ORG and by modifying XXXXXXH.

Assembling and Linking the Foreground Monitor with MCC68K

The foreground monitor can be assembled and linked with the Microtec Assembler/Linker.

To assemble the foreground monitor, enter:

```
C> asm68k -l fgmon.s > fgmon.lst
```

To link the foreground monitor, enter:

```
C> lnk68k -c fgmon.k -m -o fgmon.x > fgmon.map
```

Link command file (fgmon.k) contains:

```
format ieee
load fgmon.obj
end
```

Assembling and Linking the Foreground Monitor with AxLS

The foreground monitor can be assembled and linked with an HP Assembler/Linker.

To assemble the foreground monitor, enter:

```
as68k -L fgmon.s > fgmon.lst
```

To link the foreground monitor, enter:

```
ld68k -c fgmon.k -L > fgmon.map
```

Link command file (fgmon.k) contains:

```
name fgmon
load fgmon.o
end
```

Setting Up the Trace Vector

The foreground monitor uses the 68302 trace vector for single-stepping. Therefore, you must modify the TRACE vector (24H) in the processor's exception vector table so that it points to the TRACE_ENTRY label in the foreground monitor program.

Notes on Foreground Monitors

User Program Out of Control

A user program that runs out of control may damage the foreground monitor residing in the user memory space; if this happens, you must reload the foreground monitor. An Execution→Reset (ALT, E, E) command will automatically reload the foreground monitor.

Trace Signals and Predefined Status Values

This section describes how emulation analyzer trace signals are assigned to microprocessor address bus, data bus, and control signals.

Emulation Analyzer Trace Signals

Trace Signals	Signal Name	Signal Description
0-23	A0-A23	Address Lines 0-23
24-31	STATUS	Status Lines
32-47	D0-D15	Processor Data 0-15
48	EDMA	External DMA
49	BCLR	Bus Clear Signal
50-53	CS3-0	Chip Select Signals
54	BERR	Bus Error Signal
55	ROM	ROM Memory Access Cycle
56	GRD	Guarded Memory Access Cycle
57-60	PB11-8	Port B pins 11-8
61-63	IPL0-2	Interrupt Priority Level

Predefined Status Values

Qualifier	Status Bits (31-24)	Description
bgd	00xxx xxxxy	Emulator in background.
byte	0xxxx xxx0y	Byte cycle.
data	0xxx0 1xxxy	Data cycle.
dma	0x1xx xxxxy	Bus released to DMA device.
ext_cyc	0xxxx x0xxy	External processor cycle.
fgd	01xxx xxxxy	Emulator in foreground.
int_cyc	0xxxx x1xxy	Internal processor cycle.
intack	0xx11 1xxxy	Interrupt acknowledge cycle.
not_dma	0x0xx xxxxy	Bus not released to DMA device.
prog	0xxx1 0xxxy	Program cycle.
read	0xxxx xx1xy	Memory read.
sup	0xx1x xxxxy	Supervisor data cycle.
supdata	0xx10 1xxxy	Supervisor cycle.
supprog	0xx11 0xxxy	Supervisor program cycle.
user	0xx0x xxxxy	User cycle.
userdata	0xx00 1xxxy	User data cycle.
userprog	0xx01 0xxxy	User program cycle.
word	0xxxx xxx1y	Word cycle.
write	0xxxx xx0xy	Memory write.

Part 5

Installation Guide

Instructions for installing the product.

14

Installing the Debugger

Installing the Debugger

This chapter shows you how to install the Real-Time C Debugger.

- Requirements
- Before Installing the Debugger
- Step 1. Connect the HP 64700 to the PC
- Step 2. Install the debugger software
- Step 3. Start the debugger
- Step 4. Check the HP 64700 system firmware version
- Optimizing PC Performance for the Debugger

Requirements

- IBM compatible or NEC PC with an 80486 microprocessor and 8 megabytes of memory.
- MS Windows 3.1, set up with 20 megabytes of swap space.
- VGA Display.
- 3 Megabytes available disk space.
- Serial port, HP 64037 RS-422 port, or Novell LAN with Lan Workplace for DOS or Microsoft Lan Manager with HP ARPA Services.
- Revision A.04.00 or greater of HP 64700 system firmware. The last step in this chapter shows you how to check the firmware version number.

Before Installing the Debugger

- **Install MS Windows according to its installation manual. The Real-Time C Debugger must run under MS Windows in the 386 enhanced mode.**

  To ensure your PC is running in the 386 Enhanced Mode, double-click the PIF Editor in the Main or Accessories window. Choose the Mode pulldown in the PIF Editor menu bar. A check mark should be beside "386 Enhanced" in the Mode pulldown.

- **If the HP 64700 is to communicate with the PC via LAN:**

  Make sure the HP 64700 LAN interface is installed (see the "HP 64700 Series Installation/Service" manual).

  Install the LAN card into the PC, and install the required PC networking software.

  Obtain the Internet Address, the Gateway Address, and the Subnet Mask to be used for the HP 64700 from your Network Administrator. These three addresses are entered in integer dot notation (for example, 192.35.12.6).

- **If the HP 64700 is to communicate with the PC via RS-422:**

  Install the HP 64037 RS-422 interface card into the PC. The Real-Time C Debugger includes software that configures the RS-422 interface.

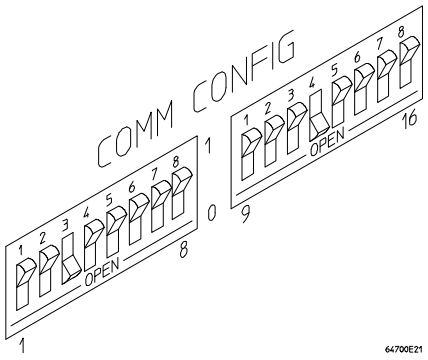
Step 1. Connect the HP 64700 to the PC

You can connect the HP 64700 to an RS-232 serial port on the PC, the Local Area Network that the PC is on, or an HP 64037 RS-422 interface that has been installed in the PC.

- To connect via RS-232
- To connect via LAN
- To connect via RS-422

To connect via RS-232

1 Set the HP 64700 configuration switches for RS-232C communication. Locate the COMM CONFIG switches on the HP 64700 rear panel, and set them as shown below.

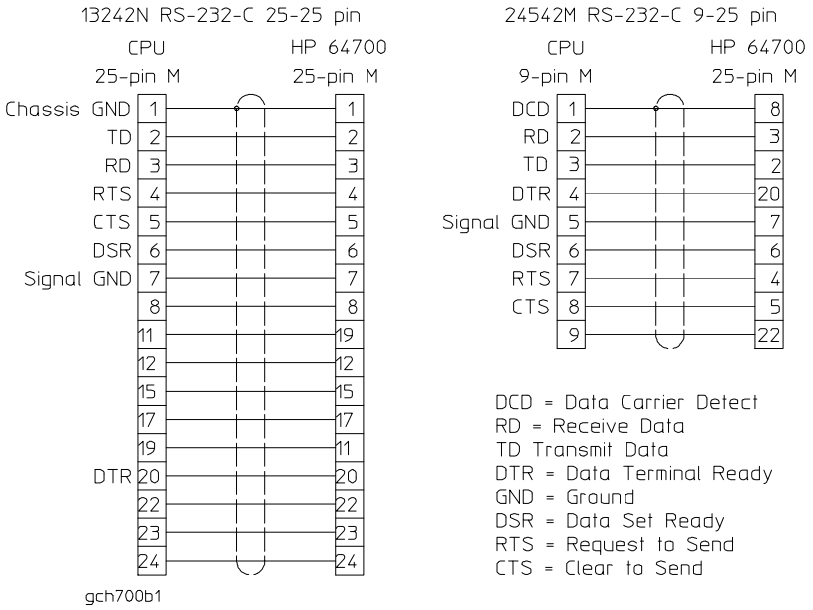


Notice that switches 1 through 3 are set to 001, respectively. This sets the baud rate to 19200.

Notice also that switches 12 and 13 are set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake which is needed to make sure all characters are processed.

2 Connect an RS-232C modem cable from the PC to the HP 64700 (for example, an HP 24542M 9-pin to 25-pin cable or an HP 13242N 25-pin to 25-pin cable).

If you want to build your own RS-232 cable, follow one of the pin-outs for HP cables shown in the following figure.



You can also use an RS-232C printer cable, but you must set HP 64700 configuration switch 4 to 1.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel. The power lamp at the lower right-hand corner of the front panel will light.

4 Start MS Windows in the 386 enhanced mode.

5 Verify RS-232 communication by using the Terminal program that is found in the Windows "Accessories" group box.

Double-click on the "Terminal" icon to open the Terminal window. Then, choose the Settings→Communications... (ALT, S, C) command, and select: 19200 Baud Rate, 8 Data Bits, 1 Stop Bit, Parity None, Hardware Flow Control, and the PC's RS-232 interface connector. Choose the OK button.

You should now be able to press the Enter key in the Terminal window to see the HP 64700's Terminal Interface prompt (for example, "R>", "M>", or "U>". The "->" prompt indicates the present firmware does not match the emulator probe, or there is no probe connected). If you see the prompt, you have verified RS-232 communication. If you do not see the prompt, refer to "If you cannot verify RS-232 communication".

If you will be using the RS-232 connection for the debugger, exit the Terminal program and go to "Step 2. Install the debugger software".

If you will be using the LAN connection, go to "To connect via LAN".

To connect via LAN

1 Set the HP 64700 LAN parameters.

If you're setting the HP 64700 LAN parameters for the first time, you must connect the HP 64700 to the PC via RS-232 before you can access the HP 64700 Terminal Interface. Follow the steps in "To connect via RS-232" and then return here.

If you're changing the LAN parameters of an HP 64700 that is already on the LAN, you can use the "telnet <HP 64700 IP address>" command to access the HP 64700 Terminal Interface.

Once the HP 64700 Terminal Interface has been accessed, display the current LAN parameters by entering the "lan" command:

```
R>lan
lan -i 15.6.25.117
lan -g 15.6.24.1
lan -s 255.255.248.0  <<- HP 64700A ONLY
lan -p 6470
Ethernet Address : 08000909BBC1
```

The "lan -i" line shows the Internet Address (or IP address). The Internet Address must be obtained from your Network Administrator. The value is entered in integer dot notation. For example, 192.35.12.6 is an Internet Address. You can change the Internet Address with the "lan -i <new IP>" command.

The "lan -g" line shows the Gateway Address which is also an Internet address and is entered in integer dot notation. This entry is optional and will default to 0.0.0.0, meaning all connections are to be made on the local network or subnet. If connections are to be made to workstations on other networks or subnets, this address must be set to the address of the gateway machine. The gateway address must be obtained from your Network Administrator. You can change the Gateway Address with the "lan -g <new gateway address>" command.

The "lan -s" line will be shown if you are using the HP 64700A, and will not be shown if you are using the HP 64700B. If this line is not shown, the Subnet Mask is automatically configured. If this line is shown, it shows the Subnet Mask in integer dot notation. This entry is optional and will default to 0.0.0.0. The default is valid only on networks that are not subnetted. (A network is

subnetted if the host portion of the Internet address is further partitioned into a subnet portion and a host portion.) If the network is subnetted, a subnet mask is required in order for the emulator to work correctly. The subnet mask should be set to all "1"s in the bits that correspond to the network and subnet portions of the Internet address and all "0"s for the host portion. The subnet mask must be obtained from your Network Administrator. You can change the Subnet Mask with the "lan -s <new subnet mask>" command .

Both the PC's subnet mask and the emulator's subnet mask must be identical unless they communicate via a gateway or a bridge. Unless your Network Administrator states otherwise, make them the same. You can check the PC's subnet mask with the "lminst" command if you are using HP-ARPA. If you are using Novell LAN WorkPlace, make sure the file \NET.CFG has the entry "ip_netmask <subnet mask>" in the section "Protocol TCPIP".

The "lan -p" line shows the base TCP service port number. The host computer interfaces communicate with the HP 64700 through two TCP service ports. The default base port number is 6470. The second port has the next higher number (default 6471). If the service port is not 6470, you must change it with the "lan -p 6470" command.

The Internet Address and any other LAN parameters you change are stored in nonvolatile memory and will take effect the next time the HP 64700 is powered off and back on again.

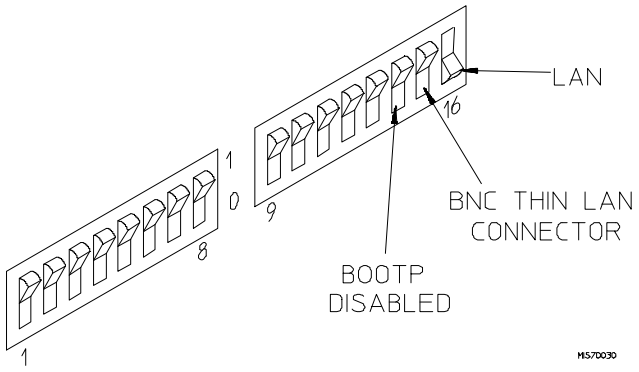
2 Exit the Terminal or telnet program.

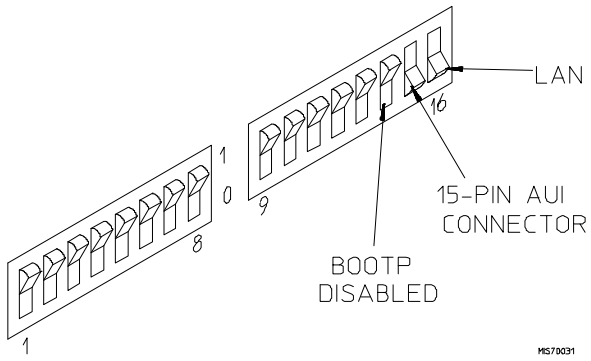
3 Turn OFF power to the HP 64700.

4 Connect the HP 64700 to the LAN. This connection can be made using either the 15-pin AUI connector or the BNC connector.

DO NOT use both connectors. The LAN interface will not work with both connected at the same time.

5 Set the HP 64700 configuration switches for LAN communication.





Switch 16 must be set to one (1) indicating that a LAN connection is being made.

Switch 15 should be zero (0) if you are connecting to the BNC connector or set to one (1) if a 15 pin AUI connection is made.

Switch 14 should be zero (0).

Set all other switches to zero (0).

6 Turn ON power to HP 64700.

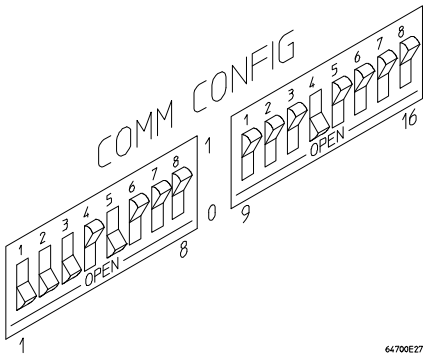
7 Verify LAN communication by using a "telnet <HP 64700 IP address>" command. This connection will give you access to the HP 64700 Terminal Interface.

You should now be able to press the Enter key in the telnet window to see the HP 64700's Terminal Interface prompt (for example, "R>", "M>", "U>", etc.). If you see the prompt, you have verified LAN communication. If you cannot connect to the HP 64700's IP address, refer to "If you cannot verify LAN communication".

To connect via RS-422

Before you can connect the HP 64700 to the PC via RS-422, the HP 64037 RS-422 Interface must have already been installed into the PC.

1 Set the HP 64700 configuration switches for RS-422 communication. Locate the COMM CONFIG switches on the HP 64700 rear panel, and set them as shown below.



Notice that switches 1 through 3 are set to 111, respectively. This sets the baud rate to 230400.

Notice that switch 5 is set to 1. This configures the 25-pin port for RS-422 communication.

Notice also that switches 12 and 13 are set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake which is needed to make sure all characters are processed.

2 Connect the 17355M cable (which comes with the HP 64037 interface) from the PC to the HP 64700.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel. The power lamp at the lower right-hand corner of the front panel will light.

If you cannot verify RS-232 communication

If the HP 64700 Terminal Interface prompt does not appear in the Terminal window:

- ☐ **Make sure that you have connected the emulator to the proper power source and that the power light is lit.**

- ☐ **Make sure that you have properly configured the data communications switches on the emulator and the data communications parameters on your controlling device. You should also verify that you are using the correct cable.**

The most common type of data communications configuration problem involves the configuration of the HP 64700 as a DCE or DTE device and the selection of the RS-232 cable. If you are using the wrong type of cable for the device selected, no prompt will be displayed.

When the RS-232 port is configured as a DCE device (S4 is set to 0), a modem cable should be used to connect the HP 64700 to the host computer of terminal. Pins 2 and 3 at one end of a modem cable are tied to pins 2 and 3 at the other end of the cable.

When the RS-232 port is configured as a DTE device (S4 is set to 1), a printer cable should be used to connect the HP 64700 to the host computer of terminal. Pins 2 and 3 at one end of a printer cable are swapped and tied to pins 3 and 2, respectively, at the other end of the cable.

If you suspect that you may have the wrong type of cable, try changing the S4 setting and turning power to the HP 64700 OFF and then ON again.

If you cannot verify LAN communication

Use the "telnet" command on the host computer to verify LAN communication. After powering up the HP 64700, it takes a minute before the HP 64700 can be recognized on the network. After a minute, try the "telnet <internet address>" command.

- If "telnet" does not make the connection:

☐ Make sure that you have connected the emulator to the proper power source and that the power light is lit.

☐ Make sure that the LAN cable is connected. Refer to your LAN documentation for testing connectivity.

☐ Make sure the HP 64700 rear panel communication configuration switches are set correctly. Switch settings are only used to set communication parameters in the HP 64700 when power is turned OFF and then ON.

☐ Make sure that the HP 64700's Internet Address is set up correctly. You must use the RS-232 port to verify this that the Internet Address is set up correctly. While accessing the emulator via the RS-232 port, run performance verification on the HP 64700's LAN interface with the "lanpv" command.

- If "telnet" makes the connection, but no Terminal Interface prompt (for example, R>, M>, U>, etc.) is supplied:

☐ It's possible that the HP 64000 software is in the process of running a command (for example, if a repetitive command was initiated from telnet in another window). You can use CTRL+c to interrupt the repetitive command and get the Terminal Interface prompt.

☐ It's also possible for there to be a problem with the HP 64700 firmware while the LAN interface is still up and running. In this case, you must turn OFF power to the HP 64700 and turn it ON again.

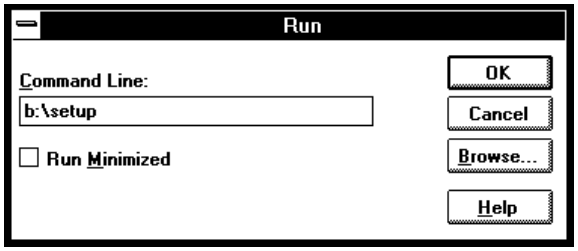
Step 2. Install the debugger software

1 If you are updating or re-installing the debugger software, you may want to save your B3621.INI file because it will be overwritten by the installation process.

2 Start MS Windows in the 386 enhanced mode.

3 Insert the 68302 REAL-TIME C DEBUGGER Disk 1 of 2 into floppy disk drive A or B.

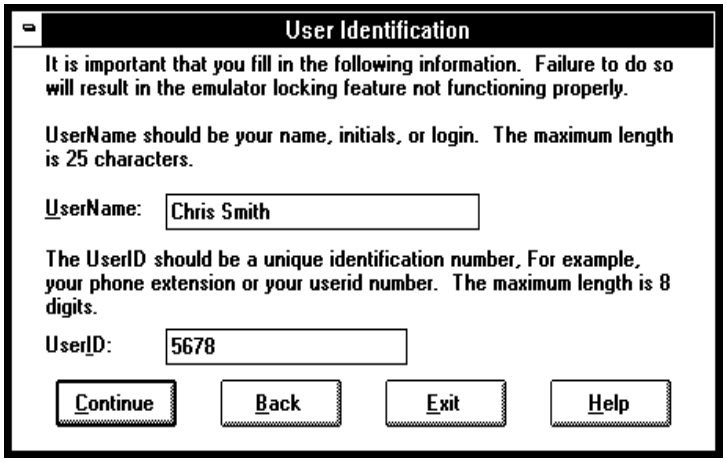
4 Choose the File→Run... (ALT, F, R) command in the Windows Program Manager. Enter "a:\setup" (or "b:\setup" if you installed the floppy disk into drive B) in the Command Line text box.



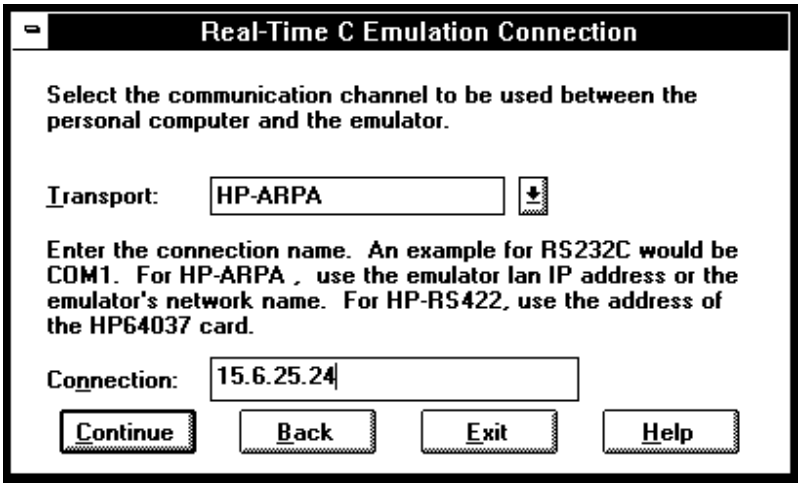
Then, choose the OK button. Follow the instructions on the screen.

You will be asked to enter the installation path. The default installation path is C:\HP\RTC\M302. The default installation path is shown wherever files are discussed in this manual.

You will be asked to enter your user ID. This information is important if the HP 64700 is on the LAN and may be accessed by other users. It tells other users who is currently using, or who has locked, the HP 64700. This information can be modified while using the Real-Time C Debugger by choosing the Settings→Communication... (ALT, S, C) command.



You will be asked to select the type of connection to be made to the HP 64700. This information can be modified while using the Real-Time C Debugger by choosing the Settings→Communication... (ALT, S, C) command.



When using the HP-RS422 transport, the connection name is the I/O address you want to use for the HP 64037 card. Enter a hexadecimal number from 100H through 3F8H, ending in 0 or 8, that does not conflict with other cards in your PC.

After you have specified the type of connection, files will be copied to your hard disk. (The B3621.TMP and B3621.HLP files are larger than most of the other files and take longer to copy.) Fill out your registration information while waiting for the files to be copied.

If the Setup program detects that one or more of the files it needs to install are currently in use by Windows, a dialog box informs you that Windows must be restarted. You can either choose to restart Windows or not. If you don't choose to restart Windows, you can either run the _MSSETUP.BAT batch file (in the same directory that the debugger software is installed in) after you have exited Windows or reinstall the debugger software later when you are able to restart Windows.

Step 3. Start the debugger

1 If the "HP Real-Time C Debugger" group box is not opened, open it by double-clicking in the icon.

2 Double-click the "M68302 Real-Time C Debugger" icon.

If you have problems connecting to the HP 64700, refer to:

- If you have RS-232 connection problems
- If you have LAN connection problems
- If you have RS-422 connection problems

If you have RS-232 connection problems

☐ Remember that Windows 3.1 only allows two active RS-232 connections at a time. To be warned when you violate this restriction, choose Always Warn in the Device Contention group box under 386 Enhanced in the Control Panel.

☐ Use the "Terminal" program (usually found in the Accessories windows program group) and set up the "Communications..." settings as follows:

```
Baud Rate: 19200 (or whatever you have chosen for the emulator)
Data Bits: 8
Parity: None
Flow Control: Hardware
Stop Bits: 1
```

When you are connected, hit the Enter key. You should get a prompt back. If nothing echos back, check the switch settings on the back of the emulator.

Switches 1 thru 3 set the baud rate as follows:

```
S1  S2  S3
0   0   0   9600
0   0   1   19200
0   1   0   2400
```

Switches 12 and 13 must be set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake, which is needed to make sure all characters are processed.

All other switches should be in the "0" position, especially switch 16 on the HP 64700 (which selects LAN/Serial interface).

Remember that if you change any of the switch positions, you must turn OFF power to the HP 64700 and turn it ON again before the changes will take effect.

- If the switches are in the correct position and you still do not get a prompt when you press return, check the following:

☐ Turn off power to the HP 64700 and then turn it on again. Press return to see if you get a prompt.

☐ Check to make sure the RS-232 cable is connected to the correct port on your PC, and that the cable is appropriate for connecting the PC to a DCE device. If the cable is intended to connect the PC to a DTE device, set switch 4 to "1" (which makes the emulator a DTE device), turn OFF power to the HP 64700, turn power ON, and try again.

☐ Check to make sure your RS-232 cable has the RTS, CTS, DSR, DCD, and DTR pins supported. If your PC RS-232 connection is a 9-pin male connection, HP cable number 24542M will work (set switch 4 to 0 if you use this cable). If your PC has a 25-pin RS-232 connector, HP cable number 13242N will work (set switch 4 to 0).

- If you wish to build your own RS-232 cable, refer to "To connect via RS-232" in the paragraph titled, "Step 1. Connect the HP 64000 to the PC" earlier in this chapter.

- When using certain RS-232 cards, connecting to an RS-232 port where the HP 64700 is turned OFF (or not connected) will halt operation of the PC. The only way to restore operation is to reboot the PC. Therefore, HP recommends you always turn ON the HP 64700 before attempting to connect via RS-232.

- If RTC reports overrun errors or simply times out, RTC may be overrunning the serial interface. In this case, try the following:

☐ Stop all unnecessary TSR's and other applications to allow the processor to service the serial interface more often.

☐ Overrun errors may occur when the serial interface card is not sufficiently buffered. Check to make sure your serial interface card uses the 16550AF UART, or better. Use the DOS command, "MSD", and when the window opens, select "COM Ports..." to see the UART chip used in your serial interface card.

If you have LAN connection problems

☐ Try to "ping" the emulator:

```
ping <hostname or IP address>
```

☐ If the emulator does not respond:

- Check that switch 16 on the emulator is "1" (emulator is attached to LAN, not RS-232 or RS-422).
- Check that switch 15 on the emulator is in the correct position for your LAN interface (either the AUI or the BNC).

Remember, if you change any switch settings on the emulator, the changes do not take effect until you turn OFF emulator power and turn it ON again.

☐ If the emulator still does not respond to a "ping," you need to verify the IP address and subnet mask of the HP 64700. To do this, connect the HP 64700 to a terminal (or to the Terminal application on the PC), change the emulator's switch settings so it is connected to RS-232, and enter the "lan" command. The output looks something like this:

```
lan -i 15.6.25.117
lan -g 15.6.24.1
lan -s 255.255.248.0
lan -p 6470
Ethernet Address : 08000909BBC1
```

The important outputs (as far as connecting) are:

"lan -i"; this shows the internet address is 15.6.25.117 in this case. If the Internet address (IP) is not what you expect, you can change it with the 'lan -i <new IP>' command.

"lan -s"; shows the subnet mask is 255.255.248 (the upper 21 bits -- 255.255.248.0 == FF.FF.F8.0). If the subnet mask is not what you expect, you can change it with the 'lan -s <new subnet mask>' command.

"lan -p"; shows the port is 6470. If the port is not 6470, you must change it with the "lan -p 6470" command.

Both the PC's subnet mask and the emulator's subnet mask must be identical unless they communicate via a gateway or a bridge. Unless your Network

Administrator states otherwise, make them the same. If you are using HP-ARPA, you can check the PC's subnet mask with the "lminst" command in a DOS window. If you are using Novell LAN WorkPlace, make sure the file \\NET.CFG has the entry "ip_netmask <subnet mask>" in the section "Protocol TCPIP." If you are using Windows for Workgroups, you can check the PC's subnet mask by looking in the [TCPIP] section of the PROTOCOL.INI file or by looking in the Microsoft TCP/IP Configuration dialog box. If you are using WINSOCK, refer to your LAN software documentation for subnet mask information.

- ☐ Occasionally the emulator or the PC will "lock up" the LAN due to excessive network traffic. If this happens, all you can do is turn OFF power to the HP 64700 or PC and turn it back ON, again. If this happens two frequently, you can try placing a gateway between the emulator/PC and the rest of your network.

If you have RS-422 connection problems

☐ Make sure the HP 64700 switch settings match the baud rate chosen when attempting the connection.

Switches 1 thru 3 set the baud rate as follows:

S1	S2	S3	
1	1	1	230400
1	1	0	115200
1	0	1	38400
1	0	0	57600
0	1	1	1200
0	1	0	2400
0	0	1	19200
0	0	0	9600

Switch 5 must be set to 1 to configure the HP 64700 for RS-422 communication.

Switches 12 and 13 must be set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake, which is needed to make sure all characters are processed.

All other switches should be in the "0" position, especially the switch that determines LAN/Serial interface (switch 16 on HP 64700).

Remember that if you change any of the switch positions, you must turn OFF power to the HP 64700 and turn it ON again before the changes will take effect.

☐ If the switches are in the correct position and you still do not get a prompt when you hit return, try turning OFF the power to the HP 64700 and tuning it ON again.

☐ If you still don't get a prompt, make sure the HP 17355M RS-422 cable is connected to the correct port on your PC.

Step 4. Check the HP 64700 system firmware version

- Choose the Help→About Debugger/Emulator... (ALT, H, D) command.

  The version information under HP 64700 Series Emulation System must show A.04.00 or greater. If the version number is less than A.04.00, you must update your HP 64700 system firmware as described in the Installing/Updating HP 64700 Firmware chapter.

Optimizing PC Performance for the Debugger

The Real-Time C Debugger is a memory and I/O intensive Windows program. Slow user interface performance may be caused by many things:

- Underpowered PC -- The Real-Time C Debugger requires an IBM compatible or NEC PC with an 80486 class microprocessor, 8 megabytes of memory, and 20 megabytes of MS Windows swap space. Because RAM is faster than swap, performance is best when there is enough RAM to accommodate all of the Real-Time C Debugger's memory usage (which is directly related to the size of your programs and the amount of debug information in them).
- Improperly configured PC -- Windows configuration may have a very significant effect on performance. The Windows swap file settings are very important (see the Virtual Memory dialog box under 386 Enhanced in the Control Panel). The larger the swap file, the better the performance. Permanent swap has superior performance.
- Disk performance (due to Windows swap file access and Windows dialog and string resource accesses from the debugger ".EXE" file) -- The disk speed has a direct impact on performance of the Real-Time C Debugger. Use of SMARTDrive or other RAM disk or caching software will improve the performance.

Various PC performance measurement and tuning tools are commercially available. Optimizing your PC performance will improve debugger interface performance and, of course, all your other PC applications will benefit as well.

15

Installing/Updating HP 64700 Firmware

Installing/Updating HP 64700 Firmware

This chapter shows you how to install or update HP 64700 firmware.

Note

If you are using an HP 64700A, it must contain the optional Flash EPROM memory card before you can install or update HP 64700 system firmware. Flash EPROM memory is standard in the HP 64700B card cage.

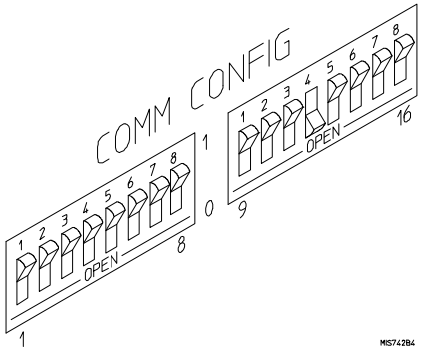
The firmware, and the program that downloads it into the HP 64700, are included with the debugger on floppy disks labeled HP 64700 EMUL/ANLY FIRMWARE.

The steps to install or update HP 64700 firmware are:

- Step 1. Connect the HP 64700 to your PC
- Step 2. Install the firmware update utility
- Step 3. Run PROGFLASH to update HP 64700 firmware
- Step 4. Verify emulator performance

Step 1. Connect the HP 64700 to the PC

1 Set the COMM CONFIG switches for RS-232C communication. To do this, locate the DIP switches on the HP 64700 rear panel, and set them as shown below.



Notice that switches 12 and 13 are set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake, which is needed to make sure all characters are processed. Switches 1, 2, and 3 are set to 0. This sets the baud rate to 9600. Switch settings are read during the HP 64700 power up routine.

2 Connect an RS-232C modem cable from the PC to the HP 64700 (for example, an HP 24542M 9-pin to 25-pin cable or an HP 13242N 25-pin to 25-pin cable).

You can also use an RS-232C printer cable, but if you do, you MUST set COMM CONFIG switch 4 to 1.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel. The power lamp at the lower right-hand corner of the front panel will light.

4 Start MS Windows in the 386 enhanced mode.

To ensure your PC is running in the 386 Enhanced Mode, double-click the PIF Editor in the Main or Accessories window. Choose the Mode pulldown in the PIF Editor menu bar. A check mark should be beside "386 Enhanced" in the Mode pulldown.

5 Verify RS-232 communication by using the Terminal program that is found in the Windows "Accessories" group box.

Double-click on the "Terminal" icon to open the Terminal window. Then, choose the Settings→Communications... (ALT, S, C) command, and select: 9600 Baud Rate, 8 Data Bits, 1 Stop Bit, Parity None, Hardware Flow Control, and the PC's RS-232 interface connector to which the RS-232 cable is attached (example: COM1). Choose the OK button.

You should now be able to press the Enter key in the Terminal window to see the HP 64700's Terminal Interface prompt (for example, p>, R>, M>, and U>. A → prompt indicates the present firmware does not match the emulator probe, or there is no probe connected). If you see the prompt, you have verified RS-232 communication. If you do not see the prompt, refer to "If you cannot verify RS-232 communication" in Chapter 14.

6 Exit the Terminal window.

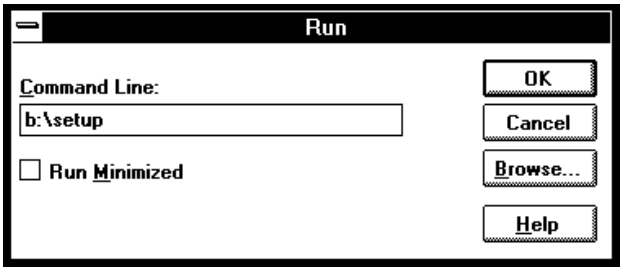
Step 2. Install the firmware update utility

The firmware update utility and emulation and analysis firmware require about 1.5 Mbytes of disk space.

1 Start MS Windows in the 386 enhanced mode.

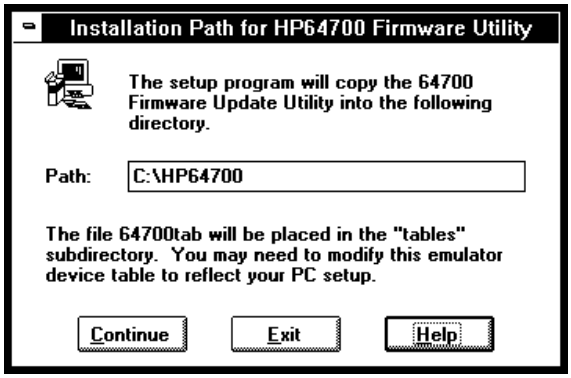
2 Insert the HP 64700 EMUL/ANLY FIRMWARE Disk 1 of 2 into floppy disk drive A or B.

3 Choose the File→Run... (ALT, F, R) command in the Windows Program Manager. Enter "a:\setup" (or "b:\setup" if you installed the floppy disk into drive B) in the Command Line text box.



Then, choose the OK button. Follow the instructions on the screen.

You will be asked to enter the installation path. The default installation path is C:\HP64700.



Wait until the Setup Exit Message dialog box appears. This indicates installation of the firmware update utility is complete.

4 After completing the installation, use the editor of your choice and edit the C:\CONFIG.SYS file to include these lines:

```
BREAK=ON
FILES=20
```

BREAK=ON allows the system to check for two break conditions: CTRL+Break, and CTRL+c.

FILES=20 allows 20 files to be accessed concurrently. This number must be at LEAST 20 to allow the firmware update utility to operate properly.

5 If you installed the files in a path other than the default (C:\HP64700), edit the C:\AUTOEXEC.BAT and C:\HP64700\BIN\FLASH.BAT files as follows:

- Edit AUTOEXEC.BAT to set the HP64700 and HPTABLES environment variables. For example:

```
SET HP64700=C:\<installation_path>
SET HPTABLES=C:\<installation_path>\TABLES
```

- Edit FLASH.BAT to identify the location of PROGFLAS.EXE. For example:

```
C:\<installation_path>\PROGFLAS.EXE
```

6 Edit the <installation_path>\TABLES\64700TAB file to indicate the communications connection you will use, as follows:

The default <installation_path>\TABLES\64700TAB file contains entries to establish the communications connection for COM1 and COM2. The content of this file is:

```
EMUL_COM1 unknown COM1 OFF 9600 NONE ON 1 8
EMUL_COM2 unknown COM2 OFF 9600 NONE ON 1 8
```

If you are using COM3 or COM4 port to update your firmware, you need to edit the <installation_path>\TABLES\64700TAB file. Either add another line or modify one of the existing lines. For example:

```
EMUL_COM3 my_emul COM3 OFF 9600 NONE ON 1 8
EMUL_COM4 unknown COM4 OFF 9600 NONE ON 1 8
```

7 Ensure the Interrupt Request Line for the selected COMx port is set to its default value. To check the default value:

1. Choose Control Panel in the Main window.
2. Choose Ports in the Control Panel window.
3. Choose the COMx port you are using and click Settings....
4. Click Advanced... in the Settings for COMx dialog box.
5. Select the default value for the Interrupt Request Line in the Advanced Settings for COMx dialog box. The default settings are:

   ```
   COM1 and COM3 = IRQ 4
   COM2 and COM4 = IRQ 3
   ```

8 Exit Windows and reboot your PC to activate the changes made to the CONFIG.SYS and AUTOEXEC.BAT files (CTRL+ALT+DEL). Installation of the firmware update utility is now complete.

Step 3. Run PROGFLASH to update HP 64700 firmware

1 Start MS Windows in the 386 enhanced mode.

2 If the "HP 64700 Firmware Utility" group box is not opened, open it by double-clicking the icon.

3 Double-click the "PROGFLASH" icon. (You can abort the PROGFLASH command by pressing CTRL+c.)

4 Enter the number that identifies the emulator you want to update. For example, enter "1" if you want to update the emulator identified by the line, "1 emul_com1 my_emul."

5 Enter the number that identifies the product whose firmware you want to update. For example, if this product is listed as number 12, enter "12":

```
Product
 1  64782
 2  E3490
 .
 .
12  647??
 .
```

6 Enter "y" to enable status messages.

The PROGFLASH command downloads code from files on the host computer into Flash EPROM memory in the HP 64700. During this download, you will see messages similar to the following:

```
Rebooting HP64700...with init -r

Downloading flash programming code:
'/hp64700/lib/npf.X'
Checking Hardware id code...
Erasing Flash ROM
Downloading ROM code: '/hp64700/update/647??.X'
  Code start 280000H
  Code size 29ABAH
Finishing up...

Rebooting HP64700...
Flash programming SUCCEEDED
```

You can display firmware version information and verify the update by choosing the Help→About Debugger/Emulator... (ALT, H, D) command in the Real-Time C Debugger.

Step 4. Verify emulator performance

- Do the performance verification procedure shown in the Installation/Service/Terminal Interface User's Guide.

Glossary

Defines terms that are used in the debugger help information.

analyzer An instrument that captures data on signals of interest at discreet periods. The emulation bus analyzer captures emulator bus cycle information synchronously with the processor's clock signal.

arm condition A condition that enables the analyzer. The analyzer is always armed unless you set the analyzer up to be armed by a signal received on the BNC port; when you do this, you can identify the arm condition in the trace specification by selecting arm in the Condition dialog boxes.

background memory A separate memory system, internal to the emulator, out of which the background monitor executes.

background monitor program An emulation monitor program that executes out of background memory.

break on trigger Causes emulator execution to break into the monitor when the trigger condition is found. This is known as a hardware breakpoint, and it lets you break on a wider variety of conditions than a software breakpoint (which replaces an opcode with a break instruction); however, depending on the speed of the processor, the actual break point may be several cycles after the one that caused the trigger.

breakpoint An address you identify in the user program where program execution is to stop. Breakpoints let you look at the state of the target system at particular points in the program.

break macro A breakpoint followed by any number of macro commands (which are the same as command file commands).

control menu The menu that is accessed by clicking the control menu box in the upper left corner of a window. You can also access control menus by pressing the "ALT" and "-" keys.

count condition Specifies whether time or the occurrences of a particular state are counted for each state in the trace buffer.

embedded microprocessor system The microprocessor system that the emulator plugs into.

emulation memory Memory provided by the emulator that can be used in place of memory in the target system.

emulation monitor A program, executed by the emulation microprocessor (as directed by the emulation system controller), that gives the emulator access to target system memory, microprocessor registers, and other target system resources.

emulator An instrument that performs just like the microprocessor it replaces, but at the same time, it gives you information about the operation of the processor. An emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory, and I/O resources.

enable condition Specifies the first condition in a two-step sequential trigger condition.

enable store condition Specifies which states get stored in the trace buffer while the analyzer searches for the enable condition.

foreground memory The memory system out of which user programs execute. Foreground memory is made up of emulation memory and target system memory.

foreground monitor program An emulation monitor program that executes out of the same memory system as user programs. This memory system is known as foreground memory and is made up of emulation memory and target system memory. The emulator only allows foreground monitor programs in emulation memory.

guarded memory Memory locations that should not be accessed by user programs. These locations are specified when mapping memory. If the user program accesses a location mapped as guarded memory, emulator execution breaks into the monitor.

macro Refers to a break macro, which is a breakpoint followed by any number of macro commands (which are the same as command file commands).

monitor A program, executed by the emulation microprocessor (as directed by the emulation system controller), that gives the emulator access to target system memory, microprocessor registers, and other target system resources.

object file An Intel OMF format absolute file that can be loaded into emulation or target system memory and executed by the debugger.

pop-up menu A menu that is accessed by clicking the right mouse button in a window.

prestore condition Specifies the states that may be stored before each normally stored state. Up to two states may be prestored for each normally stored state.

primary branch condition Specifies a condition that causes the analyzer to begin searching at another level.

restart condition Specifies the condition that restarts the two-step sequential trigger. In other words, if the restart condition occurs while the analyzer is searching for the trigger condition, the analyzer starts looking for the enable condition again.

secondary branch condition Specifies a condition that causes the analyzer to begin searching at another level. If a state satisfies both the primary and secondary branch conditions, the primary branch will be taken.

sequence levels Levels in the analyzer that let you specify a complex sequential trigger condition. For each level, the analyzer searches for primary and secondary branch conditions. You can specify a different store condition for each level. The Page button toggles the display between sequence levels 1 through 4 and sequence levels 5 through 8.

state qualifier A combination of address, data, and status values that identifies particular states captured by the analyzer.

status values Values that identify the types of microprocessor bus cycles recognized by the analyzer. You can include status values (along with

address and data values) when specifying trigger and store conditions. The status values defined for the 68302 emulator are listed under "Predefined Status Values" at the end of Chapter 13, "Concepts."

store condition Specifies which states get stored in the trace buffer.

In the "Find Then Trigger" trace set up, the store condition specifies the states that get stored after the trigger.

In the "Sequence" trace set up, each sequence level has a store condition that specifies the states that get stored while looking for the primary or secondary branch conditions.

target system The microprocessor system that the emulator plugs into.

trace state The information captured by the analyzer on a particular microprocessor bus cycle.

transfer address The program's starting address defined by the software development tools and included with the symbolic information in the object file.

trigger The captured analyzer state about which other captured states are stored. The trigger state specifies when the trace measurement is taken.

trigger condition Specifies the condition that causes states to be stored in the trace buffer.

trigger position Specifies whether the state that triggered the analyzer appear at the start, center, or end of the trace buffer. In other words, the trigger position specifies whether states are stored after, about, or before the trigger.

trigger store condition Specifies which states get stored in the trace buffer while the analyzer searches for the trigger condition.

watchpoint A variable that has been placed in the WatchPoint window where its contents can be readily displayed and modified.

Index

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING

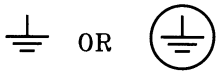
Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

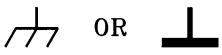
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.



rame or chassis terminal. A connection to the frame (chassis) of the quipment which normally includes all exposed metal structures.

Alternating current (power line).

Direct current (power line).

Alternating or direct current (power line).

Caution

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.