User’s Guide

Real-Time C Debugger for M37700
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.


This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

HP is a trademark of Hewlett-Packard Company.

IBM is a registered trademark of International Business Machines Corporation.

IAR is a registered trademark of IAR Inc.

MRI is a registered trademark of Microtec Research Inc.

MS and MS-DOS are registered trademarks of Microsoft Corporation.

TrueType is a registered trademark of Apple Computer, Inc.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

Windows is a trademark of Microsoft Corporation.

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

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1)(ii) of the Rights in Technical Data and Computer Software Clause at
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 B3630-97000, May 1994
Edition 2 B3630-97001, Apr 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 a 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 a 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 M37700. 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.

  Chapter 2 shows you how to use the debugger interface.
  Chapter 3 shows you how to plug the emulator into target systems.
  Chapter 4 shows you how to configure the emulator.
  Chapter 5 shows how 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 1 Quick Start Guide

1 Getting Started

Step 1. Start the debugger 26
Step 2. Setting Hardware Options 27
Step 3. Set the reset value for the stack pointer 28
Step 4. Map memory for the demo program 29
Step 5. Load the demo program 31
Step 6. Display the source file 32
Step 7. Set a breakpoint 33
Step 8. Run the demo program 34
Step 9. Delete the breakpoint 35
Step 10. Single-step one line 36
Step 11. Single-step 10 lines 37
Step 12. Display a variable 38
Step 13. Edit a variable 39
Step 14. Monitor a variable in the WatchPoint window 40
Step 15. Run until return from current function 41
Step 16. Step over a function 42
Step 17. Run the program to a specified line 43
Step 18. Display register contents 44
Step 19. Trace function flow 46
Step 20. Trace a function’s callers 47
Step 21. Trace access to a variable 48
Step 22. Exit the debugger 49
Part 2 User’s Guide

2 Using the Debugger Interface

How the Debugger Uses the Clipboard 55
Debugger Function Key Definitions 56
Starting and Exiting the Debugger 57
To start the debugger 57
To exit the debugger 58
To create an icon for a different emulator 58

Working with Debugger Windows 60
To open debugger windows 60
To copy window contents to the list file 60
To change the list file destination 61
To change the debugger window fonts 61
To set tabstops in the Source window 62

Using Command Files 63
To create a command file 63
To execute a command file 64
To create buttons that execute command files 65

3 Plugging the Emulator into Target Systems

To plug-in the HP 64146/7 emulator 69
To configure the emulator for in-circuit operation 71

4 Configuring the Emulator

Setting the Hardware Options 75
To select the emulator clock source 75
To select the processor type 77
To select the processor mode 78
To enable or disable high speed access 78
To enable or disable introduce /RDY signal 79
To enable or disable the watchdog timer 80
To enable or disable break on writes to ROM 80
5 Debugging Programs

Loading and Displaying Programs  103
To load user programs  103
To display source code only  104
To display source code mixed with assembly instructions  104
To specify nx flag for assembly instructions  105
To display source files by their names  106
To specify source file directories  107
To search for function names in the source files  108
To search for addresses in the source files  108
To search for strings in the source files  109

Displaying Symbol Information  110
To display program module information  111
To display function information  111
To display external symbol information  112
To display local symbol information  113
To display global assembler symbol information  114
To display local assembler symbol information  114
To create a user-defined symbol  115
To display user-defined symbol information  116
To delete a user-defined symbol  117
To display the symbols containing the specified string  117

Stepping, Running, and Stopping the Program  118
To step a single line or instruction  118
To step over a function  119
To step multiple lines or instructions  120
To run the program until the specified line  121
To run the program until the current function return  121
To run the program from a specified address  122
To stop program execution  122
To reset the processor  123

Using Breakpoints and Break Macros  124
To set a breakpoint  125
To disable a breakpoint  126
To delete a single breakpoint  126
To list the breakpoints and break macros  127
Setting Up Custom Trace Specifications  160
To set up a "Trigger Store" trace specification  161
To set up a "Find Then Trigger" trace specification  164
To set up a "Sequence" trace specification  168
To edit a trace specification  173
To trace "windows" of program execution  173
To store the current trace specification  175
To load a stored trace specification  176

Part 3  Reference

6  Command File and Macro Command Summary

7  Expressions in Commands

Numeric Constants   189
Symbols   190
C Operators   192

8  Menu Bar Commands

File→Load Object... (ALT, F, L)   197
File→Command Log→Log File Name... (ALT, F, C, N)   200
File→Command Log→Logging ON (ALT, F, C, O)   201
File→Command Log→Logging OFF (ALT, F, C, F)   202
File→Run Cmd File... (ALT, F, R)   203
File→Load Debug... (ALT, F, D)   205
File→Save Debug... (ALT, F, S)   206
File→Load Emulator Config... (ALT, F, E)   207
File→Save Emulator Config... (ALT, F, V)   208
File→Copy Destination... (ALT, F, P)   209
File→Exit (ALT, F, X)   210
File→Exit HW Locked (ALT, F, H)   211
File Selection Dialog Boxes   212
Execution→Run (F5), (ALT, E, U)   213
Execution→Run to Cursor (ALT, E, C)   214
Execution→Run to Caller (ALT, E, T)  215
Execution→Run... (ALT, E, R)  216
Execution→Single Step (F2), (ALT, E, N)  218
Execution→Step Over (F3), (ALT, E, O)  219
Execution→Step... (ALT, E, S)  220
Execution→Break (F4), (ALT, E, B)  223
Execution→Reset (ALT, E, E)  224
Breakpoint→Set at Cursor (ALT, B, S)  225
Breakpoint→Delete at Cursor (ALT, B, D)  226
Breakpoint→Set Macro... (ALT, B, M)  227
Breakpoint→Delete Macro (ALT, B, L)  230
Breakpoint→Edit... (ALT, B, E)  231
Variable→Edit... (ALT, V, E)  233
Variable Modify Dialog Box  235
Trace→Function Flow (ALT, T, F)  236
Trace→Function Caller... (ALT, T, C)  237
Trace→Function Statement... (ALT, T, S)  239
Trace→Variable Access... (ALT, T, V)  240
Trace→Variable Break... (ALT, T, B)  242
Trace→Edit... (ALT, T, E)  244
Trace→Trigger Store... (ALT, T, T)  245
Trace→Find Then Trigger... (ALT, T, D)  248
Trace→Sequence... (ALT, T, Q)  252
Trace→Until Halt (ALT, T, U)  256
Trace→Halt (ALT, T, H)  257
Trace→Again (F7), (ALT, T, A)  258
Condition Dialog Boxes  259
Trace Pattern Dialog Box  262
Trace Range Dialog Box  264
Sequence Number Dialog Box  266
RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D)  267
RealTime→Monitor Intrusion→Allowed (ALT, R, T, A)  268
RealTime→I/O Polling→ON (ALT, R, I, O)  269
RealTime→I/O Polling→OFF (ALT, R, I, F)  270
RealTime→Watchpoint Polling→ON (ALT, R, W, O)  271
RealTime→Watchpoint Polling→OFF (ALT, R, W, F)  272
RealTime→Memory Polling→ON (ALT, R, M, O)  273
RealTime→Memory Polling→OFF (ALT, R, M, F)  274
Assemble... (ALT, A)  275
9 Window Control Menu Commands

Common Control Menu Commands

Copy→Window (ALT, -, P, W) 319
Copy→Destination... (ALT, -, P, D) 320

Button Window Commands 321
Edit... (ALT, -, E) 321
Expression Window Commands 323
Clear (ALT, -, R) 323
Evaluate... (ALT, -, E) 324

I/O Window Commands 325
Define... (ALT, -, D) 325

Memory Window Commands 327
Display→Linear (ALT, -, D, L) 327
Display→Block (ALT, -, D, B) 328
Display→Byte (ALT, -, D, Y) 328
Display→16 Bit (ALT, -, D, 1) 328
Display→32 Bit (ALT, -, D, 3) 329
Search... (ALT, -, R) 329
Utilities→Copy... (ALT, -, U, C) 331
Utilities→Fill... (ALT, -, U, F) 332
Utilities→Image... (ALT, -, U, I) 333
Utilities→Load... (ALT, -, U, L) 335
Utilities→Store... (ALT, -, U, S) 336

Register Windows' Commands 338
Continuous Update (ALT, -, U) 338
Copy→Registers (ALT, -, P, R) 338
Register Bit Fields Dialog Box 339

Source Window Commands 341
Display→Mixed Mode (ALT, -, D, M) 341
Display→Source Only (ALT, -, D, S) 342
Display→Select Source... (ALT, -, D, L) 343
Display→Options→m0x0 344
Display→Options→m0x1 344
Display→Options→m1x0 345
Display→Options→m1x1 345
Search→String... (ALT, -, R, S) 346
Search→Function... (ALT, -, R, F) 347
Search→Address... (ALT, -, R, A) 349
Search Directories Dialog Box 350
Symbol Window Commands 351
Display→Modules (ALT, -, D, M) 351
Display→Functions (ALT, -, D, F) 352
Display→Externals (ALT, -, D, E) 352
Display→Locals... (ALT, -, D, L) 353
Display→Asm Globals (ALT, -, D, G) 354
Display→Asm Locals... (ALT, -, D, A) 355
Display→User defined (ALT, -, D, U) 357
Copy→Window (ALT, -, P, W) 357
Copy→All (ALT, -, P, A) 358
FindString→String... (ALT, -, F, S) 358
User defined→Add... (ALT, -, U, A) 359
User defined→Delete (ALT, -, U, D) 361
User defined→Delete All (ALT, -, U, L) 361

Trace Window Commands 362
Display→Bus Cycle ON (ALT, -, D, B) 362
Display→Source Only (ALT, -, D, S) 363
Display→Count→Absolute (ALT, -, D, C, A) 363
Display→Count→Relative (ALT, -, D, C, R) 364
Copy→Window (ALT, -, P, W) 364
Copy→All (ALT, -, P, A) 365
Search→Trigger (ALT, -, R, T) 365
Search→State... (ALT, -, R, S) 366
Trace Spec Copy→Specification (ALT, -, T, S) 366
Trace Spec Copy→Destination... (ALT, -, T, D) 367

WatchPoint Window Commands 368
Edit... (ALT, -, E) 368

10 Window Popup Commands

BackTrace Window Popup Commands 373
Source at Stack Level 373

Source Window Popup Commands 374
Set Breakpoint 374
Clear Breakpoint 374
11 Other Command File and Macro Commands

BEEP 379
CURSOR 380
EXIT 381
MODE SOURCE 382
MODE TRACECLOCK 383
NOP 384
WAIT 385

12 Error Messages

Part 4 Concept Guide

13 Concepts

Debugger Windows 401
The A-D Conversion Registers Window 402
The BackTrace Window 404
The Basic Register Windows 405
The Button Window 406
The D-A Conversion Registers Window 406
The Expression Window 408
The Interrupt Control Registers Window 409
The I/O Window 411
The Memory Window 412
The Mode Registers Window 413
The Other Registers Window 415
The Port Registers Window 417
The Pulse Motor Control Registers Window 419
The Source Window 421
The Status Window 423
The Symbol Window 426
Part 5 Installation Guide

14 Installing the Debugger

Requirements 455

Before Installing the Debugger 456

Step 1. Connect the HP 64700 to the PC 457
To connect via RS-232 457
To connect via LAN 460
To connect via RS-422 464
If you cannot verify RS-232 communication 465
If you cannot verify LAN communication 466
Step 2. Install the debugger software 467

Step 3. Start the debugger 470
If you have RS-232 connection problems 470
If you have LAN connection problems 472
If you have RS-422 connection problems 474

Step 4. Check the HP 64700 system firmware version 475

Optimizing PC Performance for the Debugger 476

15 Installing/Updating HP 64700 Firmware
Step 1. Connect the HP 64700 to the PC 479
Step 2. Install the firmware update utility 481
Step 3. Run PROGFLASH to update HP 64700 firmware 483
Part 1

Quick Start Guide

A few task instructions to help you get comfortable.
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. Setting the Hardware Options.
3. Set the reset value for the 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.
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 are included with the Real-Time C Debugger in the C:\HP\RTC\M7700\DEMO directory (if C:\HP\RTC\M7700 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.

Demo program is written for IAR language tools.
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 M37700 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\M7700\B3630.EXE (if C:\HP\RTC\M7700 was the installation path chosen when installing the debugger software).

3 Choose the OK button.
Step 2. Setting Hardware Options

You need to configure Hardware Options before using the emulator. The demo program runs under processor mode as proc16.

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

2. Select "proc16" in the Processor Mode group box.

3. Choose the Apply button.

4. Choose the OK button to exit.

Note: Demo program is written for IAR language tools.
Step 3. Set the reset value for the stack pointer

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

2. Enter "0ff00" in the Reset Value for Stack text box.

3. Choose the OK button.

The HP 64146/7 emulator requires the 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 stack pointer is set to the address specified.

**Note**

Breaking into the monitor from a state other than EMULATION RESET does not cause the 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

Because the emulator can use target system memory as well as emulation memory, you must tell the emulator which type of memory is to be used for particular address ranges.

The demo program reserves addresses 0h-0xffff for RAM. Map this address range as emulation memory.

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

2 Choose the Delete All button to delete current memory map.

3 Enter "0" in the Start text box.

4 Tab the cursor to the End text box and enter "0xffff".

5 Select "eram" in the Type option box.
6 Choose the Apply button.

7 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\M7700\DEMO\SAMPLE\SAMPLE.X (if C:\HP\RTC\M7700 was the installation path chosen when installing the debugger software). This program is written for IAR language tools.

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 the 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 70 in sample.c:

1. Cursor-select line 70.

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

Notice that line 70 is marked with "BP" which indicates a breakpoint has been set on the line.
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 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 70. The highlighted line indicates the current program counter.
Step 9. Delete the breakpoint

To delete the breakpoint set on line 70:

1. Cursor-select line 70.

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 31 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 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 42.
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 58.
2. Choose the Execution→Run to Cursor (ALT, E, C) command.

The program executes and stops immediately before line 58.
Step 18. Display register contents

1. Choose the Window→Register command.

   ![Basic Registers]

   The Register window opens and displays the register contents. The display is updated periodically. To run the program and see how the contents of the registers change:

2. Choose the Execution→Run (ALT, E, U) command. Or, press the F5 key.
3 To prevent the register display from being updated, choose the RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command.

Notice that register contents are replaced with "----" in the display. This shows the debugger cannot update the register display.

4 Choose the RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command to deselect the real-time mode.
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.

```
<table>
<thead>
<tr>
<th>state</th>
<th>typ</th>
<th>module</th>
<th>#line</th>
<th>function</th>
<th>source</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>SEQ</td>
<td>sample#0038</td>
<td>convert</td>
<td>c</td>
<td>1.200 uS</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SEQ</td>
<td>sample#0044</td>
<td>convert_case</td>
<td>c</td>
<td>14.40 uS</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>SEQ</td>
<td>sample#0054</td>
<td>change_status</td>
<td>c</td>
<td>232.4 uS</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>SEQ</td>
<td>sample#0054</td>
<td>next_message</td>
<td>c</td>
<td>22.60 uS</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>SEQ</td>
<td>sample#0038</td>
<td>convert</td>
<td>c</td>
<td>17.00 uS</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>SEQ</td>
<td>sample#0044</td>
<td>convert_case</td>
<td>c</td>
<td>14.40 uS</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>SEQ</td>
<td>sample#0054</td>
<td>change_status</td>
<td>c</td>
<td>323.1 uS</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>SEQ</td>
<td>sample#0062</td>
<td>next_message</td>
<td>c</td>
<td>26.80 uS</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>SEQ</td>
<td>sample#0038</td>
<td>convert</td>
<td>c</td>
<td>16.20 uS</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>SEQ</td>
<td>sample#0044</td>
<td>convert_case</td>
<td>c</td>
<td>14.40 uS</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>SEQ</td>
<td>sample#0054</td>
<td>change_status</td>
<td>c</td>
<td>254.0 uS</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>SEQ</td>
<td>sample#0062</td>
<td>next_message</td>
<td>c</td>
<td>26.80 uS</td>
</tr>
<tr>
<td>68</td>
<td>68</td>
<td>SEQ</td>
<td>sample#0038</td>
<td>convert</td>
<td>c</td>
<td>17.00 uS</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
<td>SEQ</td>
<td>sample#0044</td>
<td>convert_case</td>
<td>c</td>
<td>14.40 uS</td>
</tr>
</tbody>
</table>
```

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" on line 56 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.

<table>
<thead>
<tr>
<th>State</th>
<th>Type</th>
<th>Module/Line</th>
<th>Function</th>
<th>Source</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>PRE</td>
<td>sample\0010</td>
<td>main</td>
<td>msg</td>
<td>prestore/TG</td>
</tr>
<tr>
<td>3</td>
<td>PRE</td>
<td>sample\0011</td>
<td>next_message</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>5</td>
<td>PRE</td>
<td>sample\0011</td>
<td>main</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>6</td>
<td>PRE</td>
<td>sample\0011</td>
<td>next_message</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>8</td>
<td>PRE</td>
<td>sample\0011</td>
<td>main</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>9</td>
<td>PRE</td>
<td>sample\0011</td>
<td>next_message</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>11</td>
<td>PRE</td>
<td>sample\0011</td>
<td>main</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>12</td>
<td>PRE</td>
<td>sample\0011</td>
<td>next_message</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>14</td>
<td>PRE</td>
<td>sample\0011</td>
<td>main</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>15</td>
<td>PRE</td>
<td>sample\0011</td>
<td>next_message</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>17</td>
<td>PRE</td>
<td>sample\0011</td>
<td>main</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>18</td>
<td>PRE</td>
<td>sample\0011</td>
<td>next_message</td>
<td>msg</td>
<td>prestore</td>
</tr>
<tr>
<td>20</td>
<td>PRE</td>
<td>sample\0011</td>
<td>main</td>
<td>msg</td>
<td>prestore</td>
</tr>
</tbody>
</table>

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 71 of "main".
Step 21. Trace access to a variable

To trace access to variable "message_id":

1. Double-click "message_id" on line 70 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.
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.
Chapter 1: Getting Started

**Step 22. Exit the debugger**
User’s Guide

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

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\M7700\B3630.EXE (if C:\HP\RTC\M7700 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\M7700\B3630.EXE -T<transport> -E<connectname>" command (if C:\HP\RTC\M7700 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 B3630.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\M7700).

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 tabstops 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 tabstops in the Source window

1. Choose the Settings→Tabstops (ALT, S, T) command.

2. Select the font, font style, and size. Notice that the Sample box previews the selected font.

3. Choose the OK button.
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.
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.

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.
Plugging the Emulator into Target Systems
Plugging the Emulator into Target Systems

This chapter shows you how:

- To plug-in the HP 64146/7 emulator
- To configure the emulator for in-circuit operation

**CAUTION**
Possible Damage to the Emulation Pod. The HP 64146/7 emulator contains devices that are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the emulation pod to avoid emulator damage.

**CAUTION**
Maximum clock speed of HP 64146/7 emulators are 25MHz. These emulators do not support any operation with clock faster than 25MHz.
To plug-in the HP 64146/7 emulator

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

1. Power OFF the target system; then, power OFF the emulator.
2. Set up the switches inside the emulation pod. Refer to the manual provided with your emulation pod.
3. Remove the processor from the target system.
4. Connect the emulation pod to the target system so that pod pin 1 is inserted into target system socket pin 1 (see the figures that follow).
5. Power ON the target system; then, power ON the emulator.
6. Start the debugger.

Note: Your target system must have a clock generation circuit. The emulation pod cannot generate clock signal using a ceramic (or quartz crystal) resonator.
LCC80 Socket Connection

Connect the emulation pod so that probe pin 1 is inserted into target system socket pin 1.

SDIP64 Socket Connection

When your target system uses 64 pin shrink DIP socket, use the adaptor as shown in below, and connect the emulation pod so that pod pin 1 is inserted into adaptor socket pin 1.
To configure the emulator for in-circuit operation

The following outlines the emulator configuration for in-circuit emulation. For details on each configuration option, refer to the "Configuring the Emulator" chapter.

- Select the target system clock as the emulator clock source by choosing Settings→Emulator Config→Hardware... (ALT, S, E, H) and selecting the External option.
Chapter 3: Plugging the Emulator into Target Systems

To configure the emulator for in-circuit operation
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 the Trace Options
- Setting the Pod Configuration
- 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 select the processor type
- To select the processor mode
- To enable or disable High Speed Access
- To enable or disable introduce /RDY signal
- To enable or disable Watchdog Timer
- To enable or disable break on write to ROM
- To specify a clock speed faster than 16MHz
- To select tool type
- To select memory model

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 Hardware Config dialog box.

Selects either the emulator or target system clock.

Internal Selects the emulator clock (1,8,16,25MHz) for debugging without the target system. These frequencies can be
selected by the switches on the clock circuit board in the emulation pod.

External

Selects the target system clock for synchronous emulator-target system operation. You must use a clock input conforming to the specification for the M37700 Series microprocessor.

**Note**

Maximum clock speed of HP 64146/7 emulators are 25MHz. These emulators do not support any operation with clock faster than 25MHz.

**Note**

When the external clock is selected, your target system must have a clock generation circuit. The emulation pod cannot generate clock signal using ceramic (or quartz crystal) resonator.

**Note**

Changing the clock source selection resets the emulator.

**See Also**

To enable or disable high speed access and To specify a clock speed faster than 16MHz in the "Setting the Hardware Options" Section of the "Configuring the Emulator" chapter.
To select the processor type

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

2 Select the processor type to emulate.

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

Selecting processor type automatically sets up internal SFR/RAM/ROM area and processor mode register area to the emulator.

If your processor is not listed in supported processors, you can select "others" for this item. In this case, you need to set up proper value for internal SFR/RAM/ROM and processor mode register by yourself. (Choose the Settings→Emulator Config→Pod... (ALT, S, E, P) command.)

Note

Always set bus mode as low speed bus mode when you use M37751 emulation pod. HP 64147A emulator does not support high speed bus mode. Note that bus mode is automatically configured as high speed bus mode when you execute the emulator from user reset. Then, you need to configure bus mode as low speed bus mode.

Note

Changing the processor type resets the emulator and clears memory mapping.
To select the processor mode

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

2 Select the processor mode to emulate.

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

**Note**
Changing the processor mode resets the emulator.

To enable or disable high speed access

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

2 Select or deselect the Enable High Speed Access check box.

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

When the check box is selected:
- The emulator can access emulation memory with no wait state, up to 25MHz.
- You can map the emulation memory only to the following location.

<table>
<thead>
<tr>
<th>Memory</th>
<th>Monitor</th>
<th>Available location</th>
</tr>
</thead>
<tbody>
<tr>
<td>128K</td>
<td>Background</td>
<td>000000-01F7FF</td>
</tr>
<tr>
<td>128K</td>
<td>Foreground</td>
<td>000000-01FFFF</td>
</tr>
<tr>
<td>512K</td>
<td>Background</td>
<td>000000-07F7FF</td>
</tr>
</tbody>
</table>
When the check box is deselected:

- You can define up to 16 different map terms which can be placed wherever you like.
- The emulator can run with no wait state, up to 16MHz.
- The emulator can run with one wait state, up to 25MHz.

---

To enable or disable introduce /RDY signal

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

2. Select or deselect the Introduce /RDY input check box.

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

When the check box is selected, the emulator activates /RDY input for one clock cycle, every time the emulator accesses memory.

When the check box is deselected, the emulator inactivates /RDY input every time the emulator accesses memory.

This feature is used to run the emulator with clock faster than 16MHz. When clock is equal or slower than 16MHz, always deselect this item.
To enable or disable the watchdog timer

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

2 Select or deselect the Enable Watchdog Timer check box.

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

When the check box is selected, the watchdog timer starts counting up from reset state (FFF).

When the check box is deselected, the watchdog timer does not start counting up.

Note

Watchdog timer suspends count down while the emulator is running in background monitor.

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 Hardware 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 a clock speed faster than 16 MHz

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

2. Select or deselect the Processor Clock is Faster than 16 MHz check box.

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

When tracing microprocessor execution at speeds up to 16 MHz, time can be counted for states stored in trace memory.

However, if the processor clock speed is greater than 16 MHz, time cannot be counted. You can still count the occurrences of a qualified state.

To select tool type

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

2. Select the tool type in the Tool Type option box.

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

Note: Changing the tool type resets the emulator.
To select the memory model

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

2 Select the memory model in the Memory Model option box. Note that you need to specify appropriate compile options when you compile your program.

<table>
<thead>
<tr>
<th>Memory Model</th>
<th>small</th>
<th>medium</th>
<th>compact</th>
<th>banked</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAR</td>
<td>-ms</td>
<td>-mm</td>
<td>-mc</td>
<td>-mb</td>
</tr>
<tr>
<td>MRI</td>
<td>-Ms</td>
<td>(N/A)</td>
<td>(N/A)</td>
<td>-Ml</td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>-fNF</td>
<td>-fNF,-fFD</td>
<td>no options</td>
<td>-fFD</td>
</tr>
</tbody>
</table>

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

Note Changing the memory model resets the emulator.
Mapping Memory

This section shows you how:

- To map memory
- To map internal RAM and SFR

Because the emulator can use target memory or emulation memory (or both), it is necessary to map range of memory so that the emulator knows where to direct its access.

Up to 16 ranges of memory can be mapped, and the resolution of mapped ranges is 256 bytes (that is, memory ranges must begin on 256 byte boundaries and must be at least 256 bytes in length).

Direct memory access (DMA) to emulation memory is not permitted.

You should map all memory ranges used by your program before loading programs into 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 Select the memory type in the Type option box.

5 Choose the Apply button.

6 Repeat steps 2 through 5 for each range to be mapped.

7 Choose the Close button to exit the Memory Map dialog box.

You can map up to 16 address ranges (map terms). The minimum amount of emulation memory that can be allocated to a range is 256 bytes.

When high speed access mode is enabled in the Hardware Configuration, you can map the emulation memory only to one location, depending on the monitor type and capacity of the memory board.

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.

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

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

**Note**

The background monitor requires 2 Kbytes of emulation memory. Also, the foreground monitor requires 2 Kbytes of user program area and 1 map term.

**Example**

To map addresses 6000h through 0ffffh as an emulation RAM, 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.
To map internal RAM and SFR

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

2. Specify the starting address of internal RAM (or SFR) in the Start text box.

3. Specify the end address of internal RAM (or SFR) in the End text box.

4. Select the emulation RAM as the memory type in the Type option box.

5. Choose the Apply button.

6. Repeat steps 2 through 5 for each range to be mapped.

7. Choose the Close button to exit the Memory Map dialog box.

The HP 64146/7 emulator use internal RAM and SFR of emulation processor to emulate user program. When you direct the emulator to display the contents of internal RAM (or SFR) area, the emulator breaks to the monitor and the monitor program reads the contents of memory. Therefore, execution of user program is suspended to perform your direction.

However, you can configure the emulator so that write cycles are performed to both internal RAM (or SFR) and emulation memory. In this case, you can see the data written to emulation memory without suspending program execution.

To use this feature, you need to map these area to emulation RAM. When you do this, you can display the contents of emulation memory without suspending memory. You still can display the contents of internal RAM (or SFR) by specifying "@i" syntax in display memory operation (Refer to the To display internal RAM and SFR ).
Note: The contents of emulation memory is updated only when user program writes date to internal RAM (or SFR) area. Therefore, the contents of emulation memory may be different from the actual value of internal RAM. Especially, you should pay close attention when setting the flags of SFR.

Note: When you don’t map the internal RAM and SFR area to emulation memory, you can access the internal RAM and SFR without appending "@i".
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. Choose the OK button.

When you power up the emulator, or when you initialize it, the background monitor program is selected by default.

Reset value of stack pointer is automatically set to the end of internal RAM area by choosing processor type. When you use a processor that has no internal RAM, the stack pointer is set to FFF hex.

When you specify the reset value of stack pointer, you need to enter a 16-bit address.
Chapter 4: Configuring the Emulator

Selecting the Type of Monitor

Note
When the emulator breaks to the background monitor, the monitor program needs 5 bytes of stack.

Note
You need to reset the emulator to change the stack pointer to the reset value.

To select the foreground monitor

1. Edit the foreground monitor program source file to define its base address and processor mode register address.

2. Assemble, link and Convert 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 Stack text box. The foreground monitor requires a stack to be set up in the user program. The stack address must be an 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. Choose the OK button.
9 Use the Settings→Emulator Config→Memory Map... (ALT, S, E, M) to re-map the user program memory areas. Selecting the foreground monitor automatically resets the current memory map.

10 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.

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

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" chapter.

**Note**

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

This section shows you how:

• To enable or disable trace DMA cycles
• To enable or disable trace Refresh cycles
• To enable or disable trace Hold cycles

To enable or disable Trace DMA cycles

1 Choose the Settings→Emulator Config→Trace Options... (ALT, S, E, T) command.

2 Select or deselect Trace DMA cycles check box.

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

   When the check box is selected, each time DMA performed, one emulation analyzer state will be generated to recognize the DMA cycles.

   When the check box is deselected, no analyzer state will be generated at the occurrence of DMA. Therefore, any DMA cycles will be ignored by the analyzer.
Chapter 4: Configuring the Emulator

Setting the Trace Options

To enable or disable Trace Refresh cycles

1 Choose the Settings → Emulator Config → Trace Options... (ALT, S, E, T) command.

2 Select or deselect Trace Refresh cycles check box.

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

When the check box is selected, refresh cycles are traced by the emulation analyzer.

When the check box is deselected, refresh cycles are not traced by the emulation analyzer.

To enable or disable Trace Hold cycles

1 Choose the Settings → Emulator Config → Trace Options... (ALT, S, E, T) command.

2 Select or deselect Trace Hold cycles check box.

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

When the check box is selected, hold cycles are traced by the emulation analyzer.
Setting the pod configuration

This section shows you how:

- To select pod configuration

To select the pod configuration

1 Choose the Settings→Emulator Config→Pod... (ALT, S, E, P) command.

2 Specify the Internal SFR address in the text box.

3 Specify the Internal RAM address in the text box.

4 Specify the Internal ROM address in the text box.

5 Specify the Processor mode register address in the text box.

6 Choose the OK button.

These items are automatically set up by selecting processor type in Hardware Configuration. Therefore, you don’t need to set up these items when your processor can be specified in Hardware Configuration. If your processor is not supported in Hardware Configuration (when you select "others" for the processor configuration item), you need to set up proper value for these configuration items.

Note Changing these items resets the emulator.
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 from 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 to be fed to external devices.

To receive an arm condition input from the BNC port

• Choose the Settings→BNC→Input to Analyzer Arm (ALT, S, B, I) 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, or 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.
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.
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 specify mx flag for 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 file to be loaded.

3. Choose the Load button to load the program.

With this command, you can load IEEE-695 object file created with any of IAR, MRI or Mitsubishi programming tools for M37700.
Chapter 5: Debugging Programs

Loading and Displaying Programs

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

```
0000:000006       ( %a383 ) LDA A,03H,S
0000:000002       0x48  PHA
0000:000003       init_data();
0000:000008       JSR PG: init_data
0000:000008       while(1)
0000:000009       < convert(message_id);
0000:000006       0xac0620   LDV DT:2006H
0000:000009       0x48       CLN
0000:00000a       0x209480   JSR PG: convert
0000:00000d       message_id = next_message(message_id);
0000:00000d       0xac0620   LDV DT:2006H
0000:00000f       0x208c80   JSR PG: next_message
0000:00000f       0x80620   STR A, DT:2006H
0000:000015       0x5050   BRA 00005EH
```

See Also

To specify mx flag for assembly instruction in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

To specify mx flag for assembly instructions

1. Make the Source window the active window.

2. Specify `mx flag` to assembly instructions. Choose the
   Display→Options→m0x0, Display→Options→m0x1,
   Display→Options→m1x0 or Display→Options→m1x1 command
   from the Source window’s control menu.

When you display source code mixed with assembly instructions, you need to
tell the emulator what value of mx flag should be used to disassemble the
memory contents. This is needed because the length of operand is variable
according to mx flag.
mx flag is automatically set up when the inverse assembler encounters the following instructions.

- SEM
- CLM
- SEP X
- CLP X
- SEP M
- CLP M

**Note**
When you scroll up to see the previous lines of assembly code, top assembly line of the Source window may not be accurate.

---

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 than 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>change_status</td>
<td>800005a..80005c</td>
</tr>
<tr>
<td>convert</td>
<td>8000510..8000547</td>
</tr>
<tr>
<td>init_data</td>
<td>800048a..80004ab</td>
</tr>
<tr>
<td>main</td>
<td>800043a..8000469</td>
</tr>
<tr>
<td>next_message</td>
<td>800055c..800056f</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Externals</td>
<td></td>
</tr>
<tr>
<td>change_status</td>
<td>int ()</td>
</tr>
<tr>
<td>convert</td>
<td>void ()</td>
</tr>
<tr>
<td>convert_case</td>
<td>void ()</td>
</tr>
<tr>
<td>init_data</td>
<td>void ()</td>
</tr>
<tr>
<td>main</td>
<td>void ()</td>
</tr>
<tr>
<td>next_message</td>
<td>int ()</td>
</tr>
<tr>
<td>sample\data</td>
<td>&lt;struct&gt;</td>
</tr>
<tr>
<td>sample\data0</td>
<td>unsigned char</td>
</tr>
<tr>
<td>sample\data1</td>
<td>unsigned char</td>
</tr>
<tr>
<td>sample\message_id</td>
<td>int</td>
</tr>
<tr>
<td>sample\sample_ver</td>
<td>int</td>
</tr>
</tbody>
</table>
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 be 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

![Symbol Information Display](image)
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.

Note

The HP 64146/7 emulators do not support this function. UBROF format file compiled by IAR compiler does not include assembler local symbol information. As the result, you can not see assembler local symbols using IEEE-695 file converted from UBROF file. If you need to debug assembler local symbols, you need to specify assembler local symbols by using PUBLIC directive. In this case, you can see these symbols in global symbols.
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":

![User Defined Symbol Add dialog box](image)
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, re-display 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.

Example

When the current program counter is at line 70, 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 71.
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 B3630.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.

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 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

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.
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 43:

```c
char mes;
for (mes = &data[100]; mes != '\0'; mes++)
{
    if (*mes >= 'a' && *mes <= 'z')
        *mes -= 'a' - 'A';
    else if (*mes >= 'A' && *mes <= 'Z')
        *mes -= 'A' - 'a';
}
```

```c
int change_status(int st) /* Change status of the mesa
{
    if (st == ORIGINAL)
        return(ORIGINAL);
    else
```
To disable a breakpoint

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

2 Select the breakpoint to be disabled.

3 Select the Disable check box.

4 To close the dialog box, choose the Close button.

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

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 break macros followed by break macro commands in parentheses.

  The Breakpoint 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 Add Macro option.

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 BreakMacro Entry 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" break macros:

Position the cursor on line 33; then, choose the Breakpoint→Set Macro... (ALT, B, M) command.

Select Add Macro option.

Enter "EVALUATE *mes" 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.
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 display internal RAM and SFR
- 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 Return 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 multi-column 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 display internal RAM and SFR

**When you don’t map internal RAM (or SFR) as emulation RAM:**

- Choose the Window→Memory command.
- Double-click one of the addresses.
- Use the keyboard to enter the address of internal RAM (or SFR) to be displayed.
- Press the Return key.

Displaying internal RAM (or SFR) addresses accesses internal memory of the emulation processor. Therefore, the emulator breaks to the monitor to read the contents of memory. As the result, user program execution is suspended to display this memory.

**When you map internal RAM (or SFR) as emulation RAM:**

- Choose the Window→Memory command.
- Double-click one of the addresses.
- Use the keyboard to enter the address of internal RAM (or SFR) to be displayed.
- Press the Return key.

When you map internal RAM (or SFR) as emulation RAM, the emulator writes data to both internal RAM (or SFR) and emulation memory. In this case, you can see the data written to emulation memory without suspending program execution.

When you map internal RAM (or SFR) as emulation RAM, you can still display the contents of internal RAM (or SFR) by using the following procedure.

- Choose the Window→Memory command.
- Double-click one of the addresses.
- Specify the address of internal RAM (or SFR) with "@i".
- Press the Return key.
Chapter 5: Debugging Programs
Displaying and Editing Memory

**Note**
The contents of emulation memory is updated only when user program writes data to internal RAM or SFR. Therefore, the contents of emulation memory may be different from the actual value of internal RAM or SFR. Especially, you should pay close attention when setting the flags of SFR.

**Note**
When you modify memory, the emulator breaks to the monitor, and writes data to internal RAM or SFR. Therefore, user program execution is suspended when modifying internal RAM and SFR.

**Note**
When you direct the emulator to display the contents of internal RAM (or SFR), the emulator can display only internal RAM (or SFR) area. You can’t see and scroll-down to the other memory areas (emulation memory or target memory) from internal RAM (or SFR) areas.

**Example**
When you map internal RAM as emulation RAM, you can enter both of the following address values to display the contents of address 100 hex:

100 Accesses emulation RAM without suspending user program execution.

100@i Accesses internal RAM of emulation processor, and suspends user program execution.

**See Also**
To map internal RAM and SFR in the "Mapping Memory" Section of the "Configuring the Emulator" chapter.
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 Return 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 only 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 0fffh 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 or 16 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 2000h through 0ffffh, for the string "This":

![Search Memory Window](image)
Displaying and Editing I/O Locations

This section shows you how:

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

With the M3770 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.
To display the contents of address 2000:

Example

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 Return key.

To confirm the modified values, press the Return 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→Basic Registers command.

2 Double-click the value to be changed.

3 Use the keyboard to enter a new value.

4 Press the Return 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 Return key is pressed.

Double-clicking registers with flags or other bit fields 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

To display execution coverage

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 only 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

```c
#include
#include

int main()
{
    if(id == MESSAGE1)
        return(MESSAGE2);
    else
        return(MESSAGE1);

    void init_data()
    {
        while(id <
            convert(message_id);
            message_id = next_message(message_id);
        }
```

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 accumulated 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.

### 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.

---

**Note**

Read data from the internal RAM or SFR is not traced correctly by the emulation analyzer.

**Note**

Write data is not traced correctly, when the following conditions are met:

- The emulator is used with the M37780/81/82/83/85/95/96 emulation pods.

- The processor is operating in the memory expansion or microprocessor mode with 8 bit external bus.
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 you compile empty functions using icc7700 or nc77, stack frame is not made and you can’t trace function flow correctly.

**Example**

Function Flow Trace

<table>
<thead>
<tr>
<th>Source</th>
<th>Function</th>
<th>Line</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>convert</td>
<td></td>
<td>1.200s</td>
</tr>
<tr>
<td></td>
<td>convert_case</td>
<td></td>
<td>14.40uS</td>
</tr>
<tr>
<td></td>
<td>change_status</td>
<td></td>
<td>232.4uS</td>
</tr>
<tr>
<td></td>
<td>next_message</td>
<td></td>
<td>27.60uS</td>
</tr>
<tr>
<td></td>
<td>convert</td>
<td></td>
<td>17.00uS</td>
</tr>
<tr>
<td></td>
<td>convert_case</td>
<td></td>
<td>14.40uS</td>
</tr>
<tr>
<td></td>
<td>change_status</td>
<td></td>
<td>323.1uS</td>
</tr>
<tr>
<td></td>
<td>next_message</td>
<td></td>
<td>254.0uS</td>
</tr>
<tr>
<td></td>
<td>convert</td>
<td></td>
<td>16.20uS</td>
</tr>
<tr>
<td></td>
<td>convert_case</td>
<td></td>
<td>14.40uS</td>
</tr>
<tr>
<td></td>
<td>change_status</td>
<td></td>
<td>254.0uS</td>
</tr>
<tr>
<td></td>
<td>next_message</td>
<td></td>
<td>26.00uS</td>
</tr>
<tr>
<td></td>
<td>convert</td>
<td></td>
<td>17.00uS</td>
</tr>
<tr>
<td></td>
<td>convert_case</td>
<td></td>
<td>14.40uS</td>
</tr>
</tbody>
</table>
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.
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→Bus Cycle ON (ALT, -, D, B) 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→Bus Cycle ON (ALT, - D, B) 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 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→Bus Cycle ON (ALT, -, D, B) command.

The Display→Bus Cycle ON (ALT, -, D, B) command displays the bus cycles associated with each of 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
To display accumulated 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

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.
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.

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.

![Trace Condition](image)

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.

![Trace Pattern](image)
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. Entry into one of the other sequence levels (you specify which) will trigger the analyzer. Each level lets you specify two conditions that, when satisfied by a captured state, will cause branches to other levels:

\[
\text{if (state matches primary branch condition)} \\
\quad \text{then GOTO (level associated with primary branch)} \\
\text{else if (state matches secondary branch condition)} \\
\quad \text{then GOTO (level associated with secondary branch)} \\
\text{else} \\
\quad \text{stay at current level}
\]
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. This is different than 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; if you find the trigger condition while searching for the window disable state, you want the analyzer to branch to a sequence level that continues to search for the disable state.
Example

To trace the window of code execution between lines 32 and 37 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.
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

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

### Variable and Memory Commands

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM</td>
<td>SET</td>
<td>linenumber</td>
<td>command</td>
<td></td>
<td>Setting break macro</td>
</tr>
<tr>
<td>BM</td>
<td>SET</td>
<td>plinenum</td>
<td></td>
<td>command</td>
<td>Setting break macro</td>
</tr>
<tr>
<td>BM</td>
<td>DEL(ETE)</td>
<td>linenumber</td>
<td></td>
<td></td>
<td>Deleting break macro</td>
</tr>
<tr>
<td>BM</td>
<td>DEL(ETE)</td>
<td>plinenum</td>
<td></td>
<td></td>
<td>Deleting break macro</td>
</tr>
<tr>
<td>BP</td>
<td>SET</td>
<td>address</td>
<td></td>
<td></td>
<td>Setting breakpoint</td>
</tr>
<tr>
<td>BP</td>
<td>DEL(ETE)</td>
<td>address</td>
<td></td>
<td></td>
<td>Deleting breakpoint</td>
</tr>
<tr>
<td>BP</td>
<td>DEL(ETE)</td>
<td>ALL</td>
<td></td>
<td></td>
<td>Deleting breakpoint</td>
</tr>
<tr>
<td>BP</td>
<td>DISABLE</td>
<td>address</td>
<td></td>
<td></td>
<td>Disabling a breakpoint</td>
</tr>
<tr>
<td>BP</td>
<td>ENABLE</td>
<td>address</td>
<td></td>
<td></td>
<td>Enabling a breakpoint</td>
</tr>
<tr>
<td>EVA(LUATE)</td>
<td>address</td>
<td></td>
<td></td>
<td></td>
<td>Expression window display</td>
</tr>
<tr>
<td>EVA(LUATE)</td>
<td>&quot;strings&quot;</td>
<td></td>
<td></td>
<td></td>
<td>Printing string</td>
</tr>
<tr>
<td>EVA(LUATE)</td>
<td>CLR(AR)</td>
<td></td>
<td></td>
<td></td>
<td>Clearing Expression window</td>
</tr>
</tbody>
</table>

### Window Open/Close Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIS(PLAY)</td>
<td>BAC(KTRACE)</td>
<td></td>
<td></td>
<td></td>
<td>Opening BackTrace window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>BAS(IC)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Basic Register window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>BUT(TON)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Button window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>EXP(RESS)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Expression window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td>Opening I/O window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>MEM(ORY)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Memory window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>REG(ISTER)</td>
<td>window-num</td>
<td></td>
<td></td>
<td>Opening SFR Registers window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>SOUR(CE)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Source window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>STA(TUS)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Status window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>SYM(BOL)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Symbol window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>TRA(CE)</td>
<td></td>
<td></td>
<td></td>
<td>Opening Trace window</td>
</tr>
<tr>
<td>DIS(PLAY)</td>
<td>WAT(CHPOINT)</td>
<td></td>
<td></td>
<td></td>
<td>Opening WatchPoint window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>BAC(KTRACE)</td>
<td></td>
<td></td>
<td></td>
<td>Closing BackTrace window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>BAS(IC)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Basic Register window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>BUT(TON)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Button window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>EXP(RESS)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Expression window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>I/O</td>
<td></td>
<td></td>
<td></td>
<td>Closing I/O window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>MEM(ORY)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Memory window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>REG(ISTER)</td>
<td>window-num</td>
<td></td>
<td></td>
<td>Closing SFR Registers window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>SOUR(CE)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Source window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>STA(TUS)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Status window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>SYM(BOL)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Symbol window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>TRA(CE)</td>
<td></td>
<td></td>
<td></td>
<td>Closing Trace window</td>
</tr>
<tr>
<td>ICO(NIC)</td>
<td>WAT(CHPOINT)</td>
<td></td>
<td></td>
<td></td>
<td>Closing WatchPoint window</td>
</tr>
</tbody>
</table>
### Configuration Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MON(ITOR)</td>
<td>STA(RT) 0</td>
<td></td>
<td></td>
<td></td>
<td>Starting monitor</td>
</tr>
<tr>
<td>MON(ITOR)</td>
<td>mon-item</td>
<td>mon-ans</td>
<td></td>
<td></td>
<td>Setting up monitor</td>
</tr>
<tr>
<td>CON(Fig)</td>
<td>STA(RT) 0</td>
<td></td>
<td></td>
<td></td>
<td>Ending monitor</td>
</tr>
<tr>
<td>CON(Fig)</td>
<td>config-item</td>
<td>config-ans</td>
<td></td>
<td></td>
<td>Starting configuration</td>
</tr>
<tr>
<td>CON(Fig)</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td>Executing configuration</td>
</tr>
<tr>
<td>MAP</td>
<td>STA(RT) 0</td>
<td></td>
<td></td>
<td></td>
<td>Ending configuration</td>
</tr>
<tr>
<td>MAP</td>
<td>addr-range</td>
<td>memtype</td>
<td></td>
<td></td>
<td>Starting mapping</td>
</tr>
<tr>
<td></td>
<td>OTHER</td>
<td>memtype</td>
<td></td>
<td></td>
<td>Executing mapping</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>MNE(MONIC)</td>
<td>ON</td>
<td></td>
<td></td>
<td>Enabling Mnemonic display</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>MNE(MONIC)</td>
<td>OFF</td>
<td></td>
<td></td>
<td>Enabling Source display</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>REA(LTIME)</td>
<td>ON</td>
<td></td>
<td></td>
<td>Disabling real-time mode</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>REA(LTIME)</td>
<td>OFF</td>
<td></td>
<td></td>
<td>Enabling I/O guard</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>LOG(UARD)</td>
<td>ON</td>
<td></td>
<td></td>
<td>Disabling I/O guard</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>MEM(ORYPOLL)</td>
<td>ON</td>
<td></td>
<td></td>
<td>Enabling Memory polling</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>MEM(ORYPOLL)</td>
<td>OFF</td>
<td></td>
<td></td>
<td>Disabling Memory polling</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>WAT(CHPOLL)</td>
<td>ON</td>
<td></td>
<td></td>
<td>Enabling WatchPoint polling</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>WAT(CHPOLL)</td>
<td>OFF</td>
<td></td>
<td></td>
<td>Disabling WatchPoint polling</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>LOG</td>
<td>ON</td>
<td></td>
<td></td>
<td>Enabling log file output</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>LOG</td>
<td>OFF</td>
<td></td>
<td></td>
<td>Disabling log file output</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>BNC</td>
<td>IN</td>
<td></td>
<td></td>
<td>Setting BNC input</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>BNC</td>
<td>OUT</td>
<td></td>
<td></td>
<td>Setting BNC output</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>SOU(RCE)</td>
<td>ASK(PATH)</td>
<td></td>
<td></td>
<td>Prompt for source paths</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>SOU(RCE)</td>
<td>NOA(SKPATH)</td>
<td></td>
<td></td>
<td>Don’t prompt for source paths</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>SOU(RCE)</td>
<td>DIS(PLAY)  m0x0</td>
<td></td>
<td></td>
<td>Specifying m0x0 as mx flag</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>SOU(RCE)</td>
<td>DIS(PLAY)  m0x1</td>
<td></td>
<td></td>
<td>Specifying m0x1 as mx flag</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>SOU(RCE)</td>
<td>DIS(PLAY)  m1x0</td>
<td></td>
<td></td>
<td>Specifying m1x0 as mx flag</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>SOU(RCE)</td>
<td>DIS(PLAY)  m1x1</td>
<td></td>
<td></td>
<td>Specifying m1x1 as mx flag</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>TRACECLOCK</td>
<td>BACKGROUND</td>
<td></td>
<td></td>
<td>Trace background cycles</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>TRACECLOCK</td>
<td>BOTH</td>
<td></td>
<td></td>
<td>Trace all processor cycles</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>TRACECLOCK</td>
<td>USER</td>
<td></td>
<td></td>
<td>Trace user program cycles</td>
</tr>
</tbody>
</table>

### File Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIL(E)</td>
<td>SOU(RCE)</td>
<td>modulename</td>
<td></td>
<td></td>
<td>Displaying source file</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>OBJ(ECT)</td>
<td>filename</td>
<td></td>
<td></td>
<td>Loading object</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>SYM(BOL)</td>
<td>filename</td>
<td></td>
<td></td>
<td>Loading symbol</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>BIN(ARY)</td>
<td>filename</td>
<td></td>
<td></td>
<td>Loading data</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>APPEND</td>
<td>filename</td>
<td></td>
<td></td>
<td>Executing command file</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>COM(MAND)</td>
<td>filename</td>
<td></td>
<td></td>
<td>Specifying command log file</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>LOG</td>
<td>filename</td>
<td></td>
<td></td>
<td>Stores configuration to file</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>CON(FIGURATION)</td>
<td>LOA(D)</td>
<td>filename</td>
<td></td>
<td>Loads environment from file</td>
</tr>
<tr>
<td>FIL(E)</td>
<td>ENV(IRONMENT)</td>
<td>LOA(D)</td>
<td>filename</td>
<td></td>
<td>Stores environment to file</td>
</tr>
</tbody>
</table>

### Trace Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRA(CE)</td>
<td>FUN(CTION)</td>
<td>FLO(W)</td>
<td></td>
<td></td>
<td>Tracing function flow</td>
</tr>
<tr>
<td>TRA(CE)</td>
<td>FUN(CTION)</td>
<td>CALL(L)</td>
<td>funcname</td>
<td></td>
<td>Tracing function call</td>
</tr>
<tr>
<td>TRA(CE)</td>
<td>FUN(CTION)</td>
<td>STA(TEMENT)</td>
<td>funcname</td>
<td></td>
<td>Tracing statement</td>
</tr>
</tbody>
</table>
Chapter 6: Command File and Macro Command Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRA (CE)</td>
<td>VAR (IABLE)</td>
<td>ACC (ESS)</td>
<td>address</td>
<td></td>
<td>Tracing access to variable</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>VAR (IABLE)</td>
<td>BRE (AK)</td>
<td>address</td>
<td>value</td>
<td>Setting breakpoint variable</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>STO (P)</td>
<td></td>
<td></td>
<td></td>
<td>Stopping tracing</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>ALW (AYS)</td>
<td></td>
<td></td>
<td></td>
<td>Tracing until halt</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>AGA (IN)</td>
<td></td>
<td></td>
<td></td>
<td>Restarting tracing</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>SAV (E)</td>
<td>filename</td>
<td></td>
<td></td>
<td>Storing trace specification</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>LOA (D)</td>
<td>filename</td>
<td></td>
<td></td>
<td>Loading trace specification</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>CUS (TOMIZE)</td>
<td></td>
<td></td>
<td></td>
<td>Starts trace w/loaded spec.</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>DIS (PLAY)</td>
<td>SOU (RCE)</td>
<td></td>
<td></td>
<td>Enabling source display</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>DIS (PLAY)</td>
<td>BUS</td>
<td></td>
<td></td>
<td>Enabling bus display</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>DIS (PLAY)</td>
<td>ABS (GLUTE)</td>
<td></td>
<td></td>
<td>Displaying absolute time</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>DIS (PLAY)</td>
<td>REL (ATIVE)</td>
<td></td>
<td></td>
<td>Displaying relative time</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>COP (Y)</td>
<td>DISPLAY</td>
<td></td>
<td></td>
<td>Copying trace display</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>COP (Y)</td>
<td>ALL</td>
<td></td>
<td></td>
<td>Copying trace results</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>FIN (D)</td>
<td>TRI (GER)</td>
<td></td>
<td>state-num</td>
<td>Centers trigger in window</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>FIN (D)</td>
<td>STA (TE)</td>
<td>state-num</td>
<td></td>
<td>Centers state in window</td>
</tr>
<tr>
<td>TRA (CE)</td>
<td>COP (Y)</td>
<td>SPE (C)</td>
<td></td>
<td></td>
<td>Copying specification</td>
</tr>
</tbody>
</table>

**Symbol Window Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM (BOL)</td>
<td>LIS (T)</td>
<td>MOD (ULE)</td>
<td></td>
<td></td>
<td>Displaying module</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>LIS (T)</td>
<td>FUN (TION)</td>
<td></td>
<td></td>
<td>Displaying function</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>LIS (T)</td>
<td>EXT (ERNAL)</td>
<td></td>
<td></td>
<td>Displaying global symbol</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>LIS (T)</td>
<td>INT (ERNAL)</td>
<td>funcname</td>
<td></td>
<td>Displaying local symbol</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>LIS (T)</td>
<td>GLO (BAL)</td>
<td></td>
<td></td>
<td>Displaying global asm symbol</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>LIS (T)</td>
<td>LOC (AL)</td>
<td>modulename</td>
<td></td>
<td>Displaying local asm symbol</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>ADD</td>
<td>usersymbol</td>
<td>address</td>
<td></td>
<td>Adding user-defined symbol</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>DEL (ETE)</td>
<td>usersymbol</td>
<td></td>
<td></td>
<td>Deleting user-defined symbol</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>DEL (ETE)</td>
<td>ALL</td>
<td></td>
<td></td>
<td>Deleting all user symbols</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>MAT (CH)</td>
<td>&quot;strings&quot;</td>
<td></td>
<td></td>
<td>Displaying matched string</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>COP (Y)</td>
<td>DIS (PLAY)</td>
<td></td>
<td></td>
<td>Copying symbol display</td>
</tr>
<tr>
<td>SYM (BOL)</td>
<td>COP (Y)</td>
<td>ALL</td>
<td></td>
<td></td>
<td>Copying all symbols</td>
</tr>
</tbody>
</table>
### Command File Control Command

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

### Miscellaneous Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Param_1</th>
<th>Param_2</th>
<th>Param_3</th>
<th>Param_4</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>address</td>
<td>user_symbol</td>
<td>&quot;inst_string&quot;</td>
<td></td>
<td>In-line assembler</td>
</tr>
<tr>
<td>BEE(P)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sounding beep</td>
</tr>
<tr>
<td>BUTTON</td>
<td>label</td>
<td>&quot;command&quot;</td>
<td></td>
<td></td>
<td>Adds button to Button Window</td>
</tr>
<tr>
<td>QUI(T)</td>
<td>TO</td>
<td>filename</td>
<td></td>
<td></td>
<td>Exiting debugger</td>
</tr>
<tr>
<td>COPY</td>
<td>file-name</td>
<td></td>
<td></td>
<td></td>
<td>Specifying copy destination</td>
</tr>
<tr>
<td>COPY</td>
<td>BAS(IC)</td>
<td></td>
<td></td>
<td></td>
<td>Copying Basic Register Window</td>
</tr>
<tr>
<td>COPY</td>
<td>SOU(RCE)</td>
<td></td>
<td></td>
<td></td>
<td>Copying Source window</td>
</tr>
<tr>
<td>COPY</td>
<td>REG(ISTER)</td>
<td>window-num</td>
<td></td>
<td></td>
<td>Copying SFR Registers Window</td>
</tr>
<tr>
<td>COPY</td>
<td>MEM(ORY)</td>
<td></td>
<td></td>
<td></td>
<td>Copying Memory window</td>
</tr>
<tr>
<td>COPY</td>
<td>WAT(CHPOINT)</td>
<td></td>
<td></td>
<td></td>
<td>Copying WatchPoint window</td>
</tr>
<tr>
<td>COPY</td>
<td>BAC(KTRACE)</td>
<td></td>
<td></td>
<td></td>
<td>Copying BackTrace window</td>
</tr>
<tr>
<td>COPY</td>
<td>IO</td>
<td></td>
<td></td>
<td></td>
<td>Copying I/O window</td>
</tr>
<tr>
<td>COPY</td>
<td>EXP(RESSION)</td>
<td></td>
<td></td>
<td></td>
<td>Calling Expression window</td>
</tr>
<tr>
<td>CURSOR</td>
<td>address</td>
<td></td>
<td></td>
<td></td>
<td>Positioning cursor</td>
</tr>
<tr>
<td>DIRECTORY</td>
<td>directoryname</td>
<td></td>
<td></td>
<td></td>
<td>Directory for source search</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Non-operative</td>
</tr>
<tr>
<td>Register</td>
<td>regname</td>
<td>TO</td>
<td>value</td>
<td></td>
<td>Editing register contents</td>
</tr>
<tr>
<td>SEA(RCH)</td>
<td>STR(ING)</td>
<td>direction</td>
<td>case</td>
<td>strings</td>
<td>Searching string</td>
</tr>
<tr>
<td>SEA(RCH)</td>
<td>FUN(CTION)</td>
<td>funcname</td>
<td></td>
<td></td>
<td>Selecting function</td>
</tr>
<tr>
<td>SEA(RCH)</td>
<td>MEM(ORY)</td>
<td>size</td>
<td>addr-range</td>
<td>value</td>
<td>Searching memory</td>
</tr>
<tr>
<td>SEA(RCH)</td>
<td>MEM(ORY)</td>
<td>STR(ING)</td>
<td>&quot;strings&quot;</td>
<td></td>
<td>Searching memory for string</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>Address range</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>addr-range</td>
<td>Address range</td>
<td>Commands listed in the &quot;Reference&quot;.</td>
</tr>
<tr>
<td>case</td>
<td>Case sensing</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>command</td>
<td>Macro command</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>config-ans</td>
<td>Setting</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>config-item</td>
<td>Configuration</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>count</td>
<td>Count</td>
<td>Decimal notation</td>
</tr>
<tr>
<td>direction</td>
<td>Search direction</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>directoryname</td>
<td>Directory name</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>filename</td>
<td>File name</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>format</td>
<td>Memory file format</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>funcname</td>
<td>Function name</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>label</td>
<td>Button label</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>linenum</td>
<td>Line number</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>memtype</td>
<td>Memory type</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>modulename</td>
<td>Module name</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>mon-ans</td>
<td>Setting</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>mon-item</td>
<td>Configuration</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>plinenum</td>
<td>Macro line number</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>seconds</td>
<td>Time in seconds</td>
<td>line number.macro number (ex. 34.1)</td>
</tr>
<tr>
<td>size</td>
<td>Data size</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>space</td>
<td>Memory or I/O space</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>regname</td>
<td>Register name</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>strings</td>
<td>String</td>
<td>&quot;string&quot;</td>
</tr>
<tr>
<td>usersymbol</td>
<td>User-defined symbol</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>value</td>
<td>Value</td>
<td>See &quot;Reference&quot;.</td>
</tr>
<tr>
<td>window-num</td>
<td>SFR Window number</td>
<td>See &quot;Reference&quot;.</td>
</tr>
</tbody>
</table>
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).
- C operators (pointers, arrays, structures, unions, unary minus operators) and parentheses (specifying the order of operator evaluation).
Numeric Constants

All numeric constants without a suffix that indicates the base are assumed to be hexadecimal, except when the number refers to a count; count values are assumed to be decimal.

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, 0xFFFF0000h).
- 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, 0xFFFF0000).

**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).
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
data[0].message,s
*dat->message,s
dat->message,x32
sample\data[1].status,d32
&data[0]
```
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**: '-'
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→Disallow (ALT, R, T, D)  
• RealTime→Monitor Intrusion→Allow (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→Emulator Config→Trace Options... (ALT, S, E, T)  
• Settings→Emulator Config→Pod... (ALT, S, E, P)  
• Settings→Communication... (ALT, S, C)  
• Settings→BNC→Output Analyzer Trigger (ALT, S, B, O)  
• Settings→BNC→Inputs 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→Tabstops... (ALT, S, T)  
• Settings→Extended Settings→Trace Cycles→User (ALT, S, X, T, U)  
• Settings→Extended Settings→Trace Cycles→Monitor (ALT, S, X, T, M)  
• Settings→Extended Settings→Trace Cycles→Both (ALT, S, X, T, B)  
• Settings→Extended Settings→Load Error Abort→ON (ALT, S, X, L, O)  
• Settings→Extended Settings→Load Error Abort→OFF (ALT, S, X, L, F)
Chapter 8: Menu Bar Commands

- Settings → Extended Settings → Source Path Query → ON (ALT, S, X, S, O)
- Settings → Extended Settings → 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:

IAR 7700 Compiler
IAR 7700 Assembler
IAR UBROF IEEE-695 file converter
MRI MCC77 Compiler
MRI ASMM77 Assembler
Mitsubishi NC77 Compiler
Mitsubishi RASMC77 Assembler
Mitsubishi IEEE-695 object format converter

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

![Load Object File Dialog Box](image)
Chapter 8: Menu Bar Commands

FILE→Load Object... (ALT, F, L)

- **Current**
  - Shows the currently loaded object file.

- **File Name**
  - Specifies the object file to be loaded. The system defaults the file extension to ".x".

- **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 re-entered 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`
  - Loads the specified object file and symbols into the debugger.

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

- `FIL(E) BIN(ARY) file_name`
  - Loads only the program code from the specified object file.
FIL(E) APP(END) file_name
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:

![Command File Execution Dialog Box](image)
Chapter 8: Menu Bar Commands

File→Run Cmd File... (ALT, F, R)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Lets you enter the name of the command file to be executed.</td>
</tr>
<tr>
<td>Directory</td>
<td>Shows the current directory and the command files in that directory. You can select the command file name from this list.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Lets you specify up to five parameters that replace placeholders $1 through $5 in the command file. Parameters must be separated by blank spaces.</td>
</tr>
<tr>
<td>Executing</td>
<td>Shows the command being executed.</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes the command file.</td>
</tr>
<tr>
<td>Stop</td>
<td>Stops command file execution.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the dialog box.</td>
</tr>
<tr>
<td>Browse...</td>
<td>Opens a file selection dialog box from which you can select the command file name.</td>
</tr>
</tbody>
</table>

**Command File Command**

FILE(COMMAND) 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**

```
FILE ENVIRONMENT LOAD 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**

```
FILE ENVIRONMENT SAVE 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 re-loading the symbolic information with the File→Load Object... (ALT, F, L) command.

**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 B3630.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 this 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 is waiting for a RESET signal from target system. User program execution starts on reception of the RESET signal.
Address

Lets you enter the address from which to run.

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. In these cases, the entire function or subroutine is executed.

**Note**
The Execution→Step Over (ALT, E, O) command may fail in single-stepping the source lines containing such loop statements as "while", "for", or "do while" statements.

**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  Source window information is updated every step execution. If this option is not selected, source window information is updated after all step executions are 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.

**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 Breakpoint→Set at Cursor (ALT, B, S) command replaces the original instruction at the specified address with a BRK instruction. When the emulator detects the BRK instruction, it breaks to the monitor and restores the original instruction. When the emulator detects a BRK 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**

```plaintext
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.
**Break Macro Entry Dialog Box**

Choosing the Breakpoint→Set Macro... (ALT, B, M) command opens the following dialog box:

![Breakpoint Edit dialog box](image)

- **Breakpoint Address**: Specify line number or address followed by a decimal point and the break macro line number.

- **Add Macro**: When selected, this specifies that a break macro should be included with the breakpoint.

- **Macro Command**: Specifies the command to be added to the break macro.

- **Set**: Sets the specified macro command at the location immediately preceding the specified source line or address, or sets 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**  
Disable and delete 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 the 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 Breakpoints Set list box. Breakpoints or break macros are not actually deleted until the OK button is chosen.

**Delete All**  
Deletes all the breakpoints and break macros from the Breakpoints Set list box. Breakpoints and break macros are not actually deleted until the OK button is chosen.

**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.

**Breakpoint Dialog Box**

Choosing the Breakpoint→Edit... (ALT, B, E) command opens the following dialog box:

- **Breakpoint Address**: Specify line number or address followed by a decimal point and the break macro line number.
- **Add Macro**: When selected, this specifies that a break macro should be included with the breakpoint.
- **Macro Command**: Specifies the command to be added to the break macro.
- **Set**: Sets the specified macro command at the location immediately preceding the specified source line or address, or sets 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 `{b0 Set}`, then entering the second command and choosing `{b0 Set}`, and so on. Commands execute in the order of their entry.

Global Disable and Delete All  Disable and delete 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 the 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 Breakpoints Set list box. Breakpoints or break macros are not actually deleted until the OK button is chosen.

Delete All  Deletes all the breakpoints and break macros from the Breakpoints Set list box. Breakpoints and break macros are not actually deleted until the OK button is chosen.

Close  Closes the dialog box.

Command File Command

BP ENA(BLE) address
BP DIS(ABLE) address
BP DEL(ETE) address
BP DEL(ETE) ALL

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**

`VARIABLE 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
<th>OK</th>
<th>Cancel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;mes&quot;</td>
<td>unsigned char</td>
<td>&quot;41&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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.

Assembly language functions can also be traced provided that they comply with C function call conventions.

Note

When you compile empty functions using icc7700 or nc77, stack frame is not made and you can't trace function flow correctly.

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.

Function Caller Trace Dialog Box

Choosing the Trace→Function Caller... (ALT, T, C) command opens the following dialog box:

```
Function Caller Trace

Function:  next_message

OK         Cancel

Help
```

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.

Function Statement Trace Dialog Box

Choosing the Trace→Function Statement... (ALT, T, S) command opens the following dialog box:

<table>
<thead>
<tr>
<th>Function Statement Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function: next_message</td>
</tr>
<tr>
<td>OK</td>
</tr>
<tr>
<td>Cancel</td>
</tr>
<tr>
<td>Help</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Variable Access Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variable:</strong></td>
</tr>
<tr>
<td>message_id</td>
</tr>
<tr>
<td><strong>OK</strong></td>
</tr>
<tr>
<td><strong>Cancel</strong></td>
</tr>
<tr>
<td><strong>Help</strong></td>
</tr>
</tbody>
</table>

- **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**

TRAGE VARIA ACCESS address
Chapter 8: Menu Bar Commands

**Trace**→Variable Access... (ALT, T, V)

**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:

<table>
<thead>
<tr>
<th>Variable Break Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variable:</strong></td>
</tr>
<tr>
<td>message_id</td>
</tr>
<tr>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td><strong>OK</strong></td>
</tr>
<tr>
<td><strong>Cancel</strong></td>
</tr>
<tr>
<td><strong>Help</strong></td>
</tr>
</tbody>
</table>

- **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
Chapter 8: Menu Bar Commands

**Trace**→Variable Break... (ALT, T, B)

**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: 0xx34
- Data at an odd address: 12xx

Status values identify the types of microprocessor bus cycles. You may select status values from a predefined list.

Trigger Store Trace Dialog Box

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

![Trigger Store Trace 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.
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 multi-level 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.
Sequence Trace Dialog Box

Choosing the Trace→Sequence... (ALT, T, Q) command opens the following dialog box:

```
<table>
<thead>
<tr>
<th>Sequence Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Branch to Secondary Branch to</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
```

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 pres tore 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.
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.

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.

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.

Identifies all values not in the specified range.

Identifies the condition that arms (in other words, activates) the analyzer. The analyzer can be armed by an input signal on the BNC port.

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):

\[
\begin{align*}
\text{AND} & \quad A \text{ and } B = \bar{A} \text{ or } \bar{B} \\
\text{NAND} & \quad 
\bar{(A \text{ and } B)} = \bar{A} \text{ or } \bar{B} \\
\text{OR} & \quad \text{ or/nor} \quad \text{and B = } /\bar{A} \text{ or } /B
\end{align*}
\]

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)
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 Return 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:

<table>
<thead>
<tr>
<th>Label</th>
<th>Lets you assign a user-defined symbol to the specified address.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Lets you enter the address at which the assembly language instruction will be loaded.</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Lets you enter the assembly language instruction to be assembled.</td>
</tr>
<tr>
<td>Assemble</td>
<td>Assembles the instruction in the Mnemonic text box, and loads it into memory at the specified address.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the dialog box.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Source</td>
<td>Specifies Internal or External clock as the system clock.</td>
</tr>
<tr>
<td>Processor Type</td>
<td>Specifies the processor type.</td>
</tr>
<tr>
<td>Processor Mode</td>
<td>Specifies the mode of the processor.</td>
</tr>
<tr>
<td>Enable High Speed Access</td>
<td>Enables or disables High Speed Access Mode.</td>
</tr>
<tr>
<td>Enable Watchdog Timer</td>
<td></td>
</tr>
<tr>
<td>Enable Break on Write to ROM</td>
<td></td>
</tr>
<tr>
<td>Processor Clock is Faster than 16MHz</td>
<td></td>
</tr>
<tr>
<td>Tool Type</td>
<td></td>
</tr>
<tr>
<td>Memory Model</td>
<td></td>
</tr>
</tbody>
</table>

276
**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) PROC(ESSOR) processor_name`
Selects the processor.

`CON(FIG) MOD(E) processor_mode`
Selects the processor mode.

`CON(FIG) HAC(CESS) EN(ABLE)`
Enables High Speed Access mode.
CON(FIG) HAC(CESS) DIS(ABLE)
Disables High Speed Access mode.

CON(FIG) REA(DY) EN(ABLE)
Enables /RDY input when accesses memory.

CON(FIG) REA(DY) DIS(ABLE)
Disables /RDY input when accesses memory.

CON(FIG) WAT(CHDOG) ENA(BLE)
Enables the Watchdog Timer.

CON(FIG) WAT(CHDOG) DIS(BLE)
Disables the Watchdog Timer.

CON(FIG) ROM(BREAK) EN(ABLE)
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) SPE(ED) FAS(T)
Specifies the clock speed is faster than 16 MHz.

CON(FIG) SPE(ED) SLO(W)
Specifies the clock speed is not faster than 16 MHz. Any of the above command file commands must be preceded and followed by the respective start and end commands:

CON(FIG) TOO(LTYPE) IAR
Specifies IAR language tool.

CON(FIG) TOO(LTYPE) MRI
Specifies MRI language tool.

CON(FIG) TOO(LTYPE) MIT(SUBISHI)
Specifies Mitsubishi language tool.

CON(FIG) MEM(MODEL) SMA(LL)
Specifies memory model small.

CON(FIG) MEM(MODEL) MED(IUM)
Specifies memory model medium.
CON FIG MEM MODEL COM FACT
Specifies memory model compact.

CON FIG MEM MODEL BAN KED
Specifies memory model banked.

CON FIG STAR T
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.
Maps memory ranges.

In the HP 64146/7 emulators, you can map up to 16 address ranges (map terms). The minimum amount of emulation memory that can be allocated to a range is 256 bytes.

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).

When high speed access mode is enabled in the Hardware Configuration, you can map the emulation memory only to one location, depending on the monitor type and capacity of the memory board.
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.

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  
Maps the specified address range with the specified memory type.

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

- **Background**
- **Foreground**

Reset Value for Stack Pointer

**0100**

*NOTE: Must reset processor to take effect*

Foreground Settings

- **Monitor Address**
- **Monitor Image**

*NOTE: Must reset processor to cause load of monitor file*

Type

Let's you choose between a background monitor and a foreground monitor.

Reset Value for Stack Pointer

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.

Monitor Address

Specifies the starting address of the foreground monitor program. The foreground monitor program must be linked at this address.
When using the foreground monitor, the starting address must be located on a 2 Kbyte boundary in bank0 except internal RAM area and SFR area.

Monitor Name
- Lets you enter the name of the foreground monitor object file. The default is C:\HP\RTC\M7700\FGMON\FGMON.X (if C:\HP\RTC\M7700 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) STA(CK) address**
  Specifies the value the stack pointer is set to when the emulator breaks into the monitor from the EMULATION RESET status.
MONITOR LOGATE address
Specifies the starting address of the monitor.

MONITOR FILENAME file_name
Names the foreground monitor object file.

Any of the above command file commands must be preceded and followed by
the respective start and end commands:

MONITOR START
Starts the monitor option command section.

MONITOR END
Ends the monitor option command section.

See Also
"Selecting the Type of Monitor" in the "Configuring the Emulator" chapter.
Settings→Emulator Config→Trace Options... (ALT, S, E, T)

Specifies the trace options configuration.

**Trace Options Dialog**

Box Choosing the Settings→Emulator Config→Trace Options... (ALT, S, E, T) command opens the following dialog box:

<table>
<thead>
<tr>
<th>Trace Option Config</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace DMA cycles</td>
<td>☐</td>
</tr>
<tr>
<td>Trace Refresh cycles</td>
<td>☐</td>
</tr>
<tr>
<td>Trace Hold cycles</td>
<td>☐</td>
</tr>
</tbody>
</table>

- **Trace DMA cycles**: Enables or disables trace DMA cycles.
- **Trace Refresh cycles**: Enables or disables trace Refresh cycles.
- **Trace Hold cycles**: Enables or disables trace Hold cycles.
- **OK**: Stores current modification and closes the dialog box.
- **Cancel**: Cancels the current modification and closes the dialog box.
- **Apply**: Loads the trace option configuration to the emulator.
Chapter 8: Menu Bar Commands

Settings → Emulator Config → Trace Options... (ALT, S, E, T)

**Command File Command**

CON FIG TRA CE CYCLE DMA ON/OFF
Enables or disables trace DMA cycles.

CON FIG TRA CE CYCLE REF RESH ON/OFF
Enables or disables trace Refresh cycles.

CON FIG TRA CE CYCLE HOL D ON/OFF
Enables or disables trace Hold cycles.

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.
Settings→Emulator Config→Pod... (ALT, S, E, P)

Specifies the emulation pod configuration.

Because the following configuration is automatically set up by selecting processor type in Hardware Configuration, you don’t need to set up these items when your processor can be specified in Hardware Configuration. If your processor is not supported (when you select "others" for the configuration item), you need to set up proper value in this pod configuration.

**Pod Config Dialog Box**

Choosing the Settings→Emulator Config→Pod... (ALT, S, E, P) command opens the following dialog box:

<table>
<thead>
<tr>
<th>Pod Config</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal SFR address area</td>
<td>0..07f</td>
</tr>
<tr>
<td>Internal RAM address area</td>
<td>080..27f</td>
</tr>
<tr>
<td>Internal ROM address</td>
<td>0c000..0fff</td>
</tr>
<tr>
<td>Processor mode register address</td>
<td>05e</td>
</tr>
</tbody>
</table>

Internal SFR area Specifies internal SFR address.

Internal RAM area Specifies internal RAM address.

Internal ROM area Specifies internal ROM address.
Processor mode register address

Specifies processor mode register address.

OK
Stores current modification and closes the dialog box.

Cancel
Cancels the current modification and closes the dialog box.

Apply
Loads the trace option configuration to the emulator.

Command File Command

CON(FIG) ISF(R) address_range, address_range
Maps specified address range as internal SFR area.

CON(FIG) IRA(M) address_range
Maps specified address range as internal RAM area.

CON(FIG) IRO(M) address_range
Maps specified address range as internal ROM area.

CON(FIG) IPM(R) address
Maps specified address as processor mode register.

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.
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:

![RTC Emulation Connection Dialog Box]

The top part of the dialog box shows the current communication settings.

**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 connection and unlock the emulator.

In the HP-ARPA or Novell-WP 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 editing the command line or the .INI file, but it does allow you to see the current connection and transport being used.

The command line options for connection and transport (-E and -T) take precedence over the values in the .INI file. Note that the Program Manager item set up during debugger software installation uses these command line options.
Settings→BNC→BNC Drive Trigger (ALT, S, B, D)

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:

![Logical Diagram of BNC Connection](image)

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)
```
Chapter 8: Menu Bar Commands

Settings → BNC → BNC Drive Trigger (ALT, S, B, D)

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→BNC Receive Arm (ALT, S, B, R)

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.

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.

**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.

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**

COVERAGE RESET

**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.
Chapter 8: Menu Bar Commands

**Settings** → Font... (ALT, S, F)

**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 tabstops.

Source Tab Dialog Box

Choosing the Settings→Tabstops... (ALT, S, T) command opens the following dialog box:

![Source Tab Dialog Box](image)

Tab width in source window display: Enter the number of spaces between tabstops. This also affects the tab width for source lines in the Trace window.

OK Sets the tabstops, and closes the dialog box.

Cancel Cancels tabstop setting, and closes the dialog box.

See Also

"To set tabstops in the Source window" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.
Settings→Extended Setting→Trace Cycles→User (ALT, S, X, T, U)

Trace foreground emulation microprocessor operation.
This is the normal setting.

Command File Command
MOD(E) TRA(CECLOCK) USE(R)
Trace background emulation microprocessor operation.
This is rarely a useful setting when debugging programs.

**Command File Command**

MOD (E)  TRA (CECLOCK)  BAC (KGROUND)
Settings→Extended Setting→Trace Cycles→Both
(ALT, S, X, T, B)

Trace both foreground and background emulation microprocessor operation.

**Command File Command**

MOD (E) TRA (CECLOCK) BOT (H)
An error during an object file or memory load caused 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)
Chapter 8: Menu Bar Commands

Settings→Extended Setting→Load Error Abort→Off (ALT, S, X, L, F)

Settings→Extended Setting→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 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)
Settings→Extended Setting→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) SOUR (RCE) ASK (PATH)
Settings→Extended Setting→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**

\texttt{MOD (E) SOU (RCE) NOA (SKPATH)}
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.

The debugger has the following windows:

- BackTrace
- Basic Registers
- Button
- Expression
- I/O
- Memory
- SFR Registers
- Source
- Status
- Symbol
- Trace
- WatchPoint

For details on the each of these windows, refer to the "Debugger Windows" section in the "Concepts" information.

**Command File Command**

**DIS(PLAY) window-name**

Opens the specified window.
DIS(PLAY) REG(ISTER) window-number
Opens the specified SFR registers window.

ICO(NIC) window-name
Closes the specified window.

ICO(NIC) REG(ISTER) window-number
Closes the specified SFR registers window.

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.

```
ICO(NIC) window-name
```

Closes the specified window.

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.
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
- Expression Window Commands
- I/O Window Commands
- Memory Window Commands
- Registers Windows’ 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) BAS (IC)
COP (Y) BUT (TON)
COP (Y) EXP (RESSION)
COP (Y) IO
COP (Y) MEM (ORY)
COP (Y) REG (ISTER) window-number
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**

COPY (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.

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.

You can enter command file commands.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specifies the button label to be associated with the command.</td>
</tr>
<tr>
<td>Add</td>
<td>Adds the button to the button window.</td>
</tr>
<tr>
<td>Button Definitions</td>
<td>Lists the currently defined buttons. You can select button definitions for deletion by clicking on them.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the button definition selected in the Button Definitions list box.</td>
</tr>
<tr>
<td>Delete All</td>
<td>Deletes all buttons from the Button window.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the dialog box.</td>
</tr>
</tbody>
</table>

**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.
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:

![Evaluate Expression Dialog Box](image)

- **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:

![I/O Setting 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 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 IOSPACE/MEMORY address
Registers the I/O address to be displayed in the specified size.

IO DEL(ETE) BYTE/WORD 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 multi-column 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:

<table>
<thead>
<tr>
<th>Search Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value:</td>
</tr>
<tr>
<td>String:</td>
</tr>
<tr>
<td>Start:</td>
</tr>
<tr>
<td>End:</td>
</tr>
<tr>
<td>Size:</td>
</tr>
</tbody>
</table>

Value         Lets you enter a value.
### Memory Window Commands

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String</strong></td>
<td>Lets you enter a string.</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>Lets you enter the starting address of the memory range to search.</td>
</tr>
<tr>
<td><strong>End</strong></td>
<td>Lets you enter the end address of the memory range to search.</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.</td>
</tr>
<tr>
<td><strong>Execute</strong></td>
<td>Searches for the specified value or string.</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>Closes the dialog box.</td>
</tr>
</tbody>
</table>

#### 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:

<table>
<thead>
<tr>
<th>Start:</th>
<th>00000000</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>End:</td>
<td>00000007</td>
<td>Close</td>
</tr>
<tr>
<td>Destination:</td>
<td>04000</td>
<td></td>
</tr>
<tr>
<td>Size:</td>
<td>byte</td>
<td>16 Bits</td>
</tr>
</tbody>
</table>

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
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:

![Memory Image Dialog Box]

- **Start**: Lets you enter the starting address of the memory area.
- **End**: Lets you enter the 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:

![Load Binary File 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
```
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:

![Store Binary File Dialog Box](image)

- **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
Let you enter the starting address of the memory range to be stored.

### End
Let 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

<table>
<thead>
<tr>
<th>Command File Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM(ORY) STO(RE) MOT(OSREC) addr-range filename</td>
</tr>
<tr>
<td>MEM(ORY) STO(RE) INT(ELHEX) addr-range filename</td>
</tr>
</tbody>
</table>

### 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 Windows’ Commands

This section describes the following commands:

- Continuous Update (ALT, -, U)
- Copy→Registers (ALT, -, P, R)

Continuous Update (ALT, -, U)

Specifies whether the Register window contents should be continuously updated while running programs.

A check mark (✓) next to the command shows that continuous update is active.

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)
Copy Basic Register windows contents to the destination file.

COP(Y) REG(ISTER) window-number
Copy SFR Register windows contents to the destination file.
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.

![Register Bit Fields Dialog Box](image)

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)**
- **Display→Options→m0x0 (ALT, -, D, O)**
- **Display→Options→m0x1 (ALT, -, D, O)**
- **Display→Options→m1x0 (ALT, -, D, O)**
- **Display→Options→m1x1 (ALT, -, D, O)**
- **Search→String... (ALT, -, R, S)**
- **Search→Function... (ALT, -, R, F)**
- **Search→Address... (ALT, -, R, A)**

**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
Display→Options→m0x0

Specify m0x0 as mx flag. When you display source code mixed with assembly instructions, you need to tell the emulator what value of mx flag should be used to disassemble the memory contents. This is needed because the length of operand is variable according to mx flag.

Every time the inverse assembler encounters an instruction which changes mx flag, the value set by the instruction is used to disassemble memory contents.

Command File Command

MOD(E) SOURCE DISPLAY m0x0

Display→Options→m0x1

Specify m0x1 as mx flag. When you display source code mixed with assembly instructions, you need to tell the emulator what value of mx flag should be used to disassemble the memory contents. This is needed because the length of operand is variable according to mx flag.

Every time the inverse assembler encounters an instruction which changes mx flag, the value set by the instruction is used to disassemble memory contents.

Command File Command

MOD(E) SOURCE DISPLAY m0x1
Display→Options→m1x0

Specify m1x0 as \textit{mx flag}. When you display source code mixed with assembly instructions, you need to tell the emulator what value of \textit{mx flag} should be used to disassemble the memory contents. This is needed because the length of operand is variable according to \textit{mx flag}.

Every time the inverse assembler encounters an instruction which changes \textit{mx flag}, the value set by the instruction is used to disassemble memory contents.

\textbf{Command File Command}

\texttt{MOD(E) SOURCER DISPLAY m1x0}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Source Window Commands} & 345 \\
\hline
\end{tabular}
\end{table}

Display→Options→m1x1

Specify m1x1 as \textit{mx flag}. When you display source code mixed with assembly instructions, you need to tell the emulator what value of \textit{mx flag} should be used to disassemble the memory contents. This is needed because the length of operand is variable according to \textit{mx flag}.

Every time the inverse assembler encounters an instruction which changes \textit{mx flag}, the value set by the instruction is used to disassemble memory contents.

\textbf{Command File Command}

\texttt{MOD(E) SOURCER DISPLAY m1x1}
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find What</td>
<td>Lets you enter the string.</td>
</tr>
<tr>
<td>Match Case</td>
<td>Selects or deselects case matching.</td>
</tr>
<tr>
<td>Direction</td>
<td>Specifies that the search be from the current cursor position.</td>
</tr>
<tr>
<td>Find Next</td>
<td>Searches for the string.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the dialog box.</td>
</tr>
</tbody>
</table>

Find What

Match Case

Up

Down

Find Next

Close
Command File Command

SEARCH STRING 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.
Chapter 9: Window Control Menu Commands

Source Window Commands

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

SEARCH FUNCTION 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:

![Search Address Dialog Box](image)

- **Address**: Lets you enter the address to search for.
- **Find**: Searches for the specified address.
- **Close**: Closes the dialog box.

Command File Command

SEA(RCH) ADD(RESS) address

See Also

"To search for addresses in the source files" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.
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.
- **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:

<table>
<thead>
<tr>
<th>Function:</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert_case</td>
<td>Cancel</td>
</tr>
</tbody>
</table>

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) L IST(G) 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.

Note

When you use IAR compiler, this function is not available. UBROF format file compiled by IAR compiler does not include assembler local symbol information. As the result, you cannot see assembler local symbols using IEEE-695 file converted from UBROF file. If you need to debug assembler local symbols with IAR tools, you need to specify assembler local symbols by using PUBLIC directive. In this case, you can see these symbols in global symbols.

Assembler Symbol Dialog Box

Choosing the Display→Asm Locals... (ALT, -, D, A) command opens the following dialog box:

<table>
<thead>
<tr>
<th>Module</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample</td>
<td>Cancel</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>String: init</th>
<th>OK</th>
<th>Cancel</th>
<th>Help</th>
</tr>
</thead>
</table>

String       Specifies the string.
OK           Executes the command and closes the dialog box.
Cancel  C cancels the command and closes the dialog box.

**Command File Command**

\texttt{SYM(BOL) MATCH 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→Bus Cycle ON (ALT, -, D, B)
• Display→Source Only (ALT, -, D, S)
• 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→Bus Cycle ON (ALT, -, D, B)

Selects the bus cycle mixed 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→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→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 accumulated 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 accumulated 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:

<table>
<thead>
<tr>
<th>State:</th>
<th>Find</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Enter]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Let 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.
Window Popup Commands
Window Popup Commands

This chapter describes the commands that can be chosen from the popup menus in debugger windows. Popup menus are accessed by clicking the right mouse button in the window.

- BackTrace Window Popup Commands
- Source Window Popup Commands
BackTrace Window Popup 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 Popup 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.
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
- CURSOR
- EXIT
- MODE SOURCE
- MODE TRACECLOCK
- NOP
- WAIT
BEEP

Sounds beep during command file or break macro execution.

Command File Command

BEEP
CURSOR

Locates the cursor at an address in the Source window.

This command can be used to display a particular address in the Source window or, when used before the COME command, to run to a particular address.

**Command File Command**

```text
CURSOR address
```
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.
MODE SOURCE

Specifies whether 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.

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) ASK (PATH)*

Prompts for source file paths when source files are not found. This is the default.

*MOD (E) SOU (RCE) NOA (SKPATH)*

Turns off prompting for source file paths.
MODE TRACECLOCK

Specifies tracing of foreground or background emulation microprocessor operation.

Command File Command

MOD (E) TRA (CECLOCK) BAC (KGROUND)
Traces background emulation microprocessor operation. This is rarely a useful setting when debugging programs.

MOD (E) TRA (CECLOCK) BOT (H)
Traces both background and foreground emulation microprocessor operation.

MOD (E) TRA (CECLOCK) USE (R)
Traces foreground emulation microprocessor operation. This is the normal setting.
Chapter 11: Other Command File and Macro Commands

NOP

No operation.

This command may be used to prefix comment lines in command files.

**Command File Command**

NOP

NOP comments
WAIT

Inserts wait delays during command file execution.

**Command File Command**

WAIT (T) MON (ITOR)
Waits until MONITOR status.

WAIT (T) RUN
Waits until RUN status.

WAIT (T) UNK (NOWN)
Waits until UNKNOWN status.

WAIT (T) SLO (W)
Waits until SLOW CLOCK status.

WAIT (T) TGT (RESET)
Waits until TARGET RESET status.

WAIT (T) SLE (EP)
Waits until SLEEP status.

WAIT (T) GRA (NT)
Waits until BUS GRANT status.

WAIT (T) NOB (US)
Waits until NOBUS status.

WAIT (T) TCO (M)
Waits until the trace is complete.

WAIT (T) THA (LT)
Wait until the trace is halted.

WAIT (T) TIM (E) seconds
Waits for a number of seconds.
Chapter 11: Other Command File and Macro Commands

```
WAIT
```
Error Messages
Error Messages

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 your PC has.

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 (100, 108, 110, etc.). Select an I/O address that does not conflict with the other cards in your PC.

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, do not worry about 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).

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, do not worry about 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’s 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 hasn’t 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.

Usually, you must cycle power to the HP 64700 to fix this problem.

**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 debugger on your system, this can happen when you try to execute both of them at the same time or when you execute one version then the other without restarting Windows. (Once DLLs have been loaded into Windows memory, they stay there until Windows exits.)

This can also happen if you have somehow loaded different versions of the DLLs and the executable. In this case, you must reload your software.

**Incorrect LAN Address (HP-ARPA)**

A LAN address can be one of two types: an IP address, or a host name.
Chapter 12: Error Messages

An IP address consists of 4 digits separated by dots. Example:
15.6.28.0

A hostname is a name which is related (mapped) to an IP address by a database. For example, the file \LANMAN.DOS\ETC\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.

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 about a 15-second delay 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 4 digits separated by dots. Example:
15.6.28.0

A hostname is a name which 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 NETCFG files are searched.
**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 amount 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 debugger application's '.INI' file.

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 debugger application's '.INI' file.

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 debugger application's ".INI" file.

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, 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 will correct the entry in the .INI file, but if the error is in the command line option, you need to 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 when the debugger sends data and when the LAN actually sends it. When this pool is exhausted, debugger cannot send any data across the LAN.

The size of the sockets buffer pool is configured in the network installation procedure.

**LAN communications error**

This occurs for 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 means that 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.

**No initialization (.INI) file was found**

For example, if the application is Bxxxx.EXE, the Bxxxx.INI file is expected to be found in the same directory.

To fix this problem, you can re-create the initialization file by copying information from the default file, for example BxxxxDEF.INI, which is in the same directory as the application. If you cannot find the default initialization file either, you can re-install the debugger software.

**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.

**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 thru 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.

**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 hasn’t 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 Bxxxx.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’re 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’re 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.

[TCP_XFR]
TCP_CONNTIMEOUT=<seconds>

The default connection timeout is 30 seconds.
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:

- A-D Conversion Registers
- BackTrace
- Basic Registers
- Button
- D-A Conversion Registers
- Expression
- Interrupt Control Registers
- I/O
- Memory
- Mode Registers
- Port Registers
- Pulse Motor Control Registers
- Source
- Status
- Symbol
- Timer Registers
- Trace
- UART0 Registers
- UART1 Registers
- Watchdog Timer Registers
- WatchPoint
The A-D Conversion Registers Window

The A-D Conversion Registers window displays contents of A-D Conversion Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCCRE</td>
<td>A-D control</td>
</tr>
<tr>
<td>ADSFR</td>
<td>A-D sweep pin selection</td>
</tr>
<tr>
<td>ADR0</td>
<td>A-D register 0</td>
</tr>
<tr>
<td>ADR1</td>
<td>A-D register 1</td>
</tr>
<tr>
<td>ADR2</td>
<td>A-D register 2</td>
</tr>
<tr>
<td>ADR3</td>
<td>A-D register 3</td>
</tr>
<tr>
<td>ADR4</td>
<td>A-D register 4</td>
</tr>
<tr>
<td>ADR5</td>
<td>A-D register 5</td>
</tr>
<tr>
<td>ADR6</td>
<td>A-D register 6</td>
</tr>
<tr>
<td>ADR7</td>
<td>A-D register 7</td>
</tr>
</tbody>
</table>

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.

The A-D Conversion Registers window’s 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.

A-D Conversion Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>eadcr</td>
<td>A-D control</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>adspsr</td>
<td>A-D sweep pin selection</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>adr0</td>
<td>A-D register 0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr1</td>
<td>A-D register 1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr2</td>
<td>A-D register 2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr3</td>
<td>A-D register 3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr4</td>
<td>A-D register 4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr5</td>
<td>A-D register 5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr6</td>
<td>A-D register 6</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adr6</td>
<td>A-D register 7</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The BackTrace Window

The BackTrace window displays the function associated with the current program counter value and this function’s caller functions backward. 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 Popup Commands" in the "Window Popup Commands" chapter.
The Basic Register Windows

The Basic Register windows display the contents of registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>205C</td>
<td>Accumulator A</td>
</tr>
<tr>
<td>B</td>
<td>0000</td>
<td>Accumulator B</td>
</tr>
<tr>
<td>X</td>
<td>0000</td>
<td>Index X</td>
</tr>
<tr>
<td>Y</td>
<td>0000</td>
<td>Index Y</td>
</tr>
<tr>
<td>SP</td>
<td>2251</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>DI</td>
<td>00</td>
<td>Data Bank</td>
</tr>
<tr>
<td>PG</td>
<td>00</td>
<td>Program Bank</td>
</tr>
<tr>
<td>PC</td>
<td>0011</td>
<td>Program Counter</td>
</tr>
<tr>
<td>DPR</td>
<td>205C</td>
<td>Direct Page</td>
</tr>
<tr>
<td>PS</td>
<td>0004</td>
<td>Processor Status</td>
</tr>
</tbody>
</table>

Each register is represented by a row which holds a mnemonic name, a current value, and a description of the register contents.

You can modify register contents by double-clicking on the value, using the keyboard to type in the new value, and pressing the Return key.

The Basic Registers window's 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.

Registers Window Commands in the "Window Control Menu 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.

The D-A Conversion Registers Window

The D-A Conversion Registers window displays contents of D-A Conversion Registers.

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.

The D-A Conversion Registers window’s 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.

D-A Conversion Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address Size</th>
<th>Bit Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>dar0</td>
<td>D-A register 0</td>
<td>1AH</td>
<td>1</td>
</tr>
<tr>
<td>dar1</td>
<td>D-A register 1</td>
<td>1CH</td>
<td>1</td>
</tr>
</tbody>
</table>
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 Interrupt Control Registers Window

The Interrupt Control Registers window displays contents of Interrupt Control Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCICR-00</td>
<td>A/D conversion intr ctrl</td>
<td></td>
</tr>
<tr>
<td>UART0CR-00</td>
<td>UART0 Tx intr ctrl</td>
<td></td>
</tr>
<tr>
<td>UART1CR-00</td>
<td>UART0 Rx intr ctrl</td>
<td></td>
</tr>
<tr>
<td>UART2CR-00</td>
<td>UART1 Tx intr ctrl</td>
<td></td>
</tr>
<tr>
<td>UART3CR-00</td>
<td>UART1 Rx intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TA0ICR-00</td>
<td>Timer A0 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TA1ICR-00</td>
<td>Timer A1 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TA2ICR-00</td>
<td>Timer A2 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TA3ICR-00</td>
<td>Timer A3 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TA4ICR-00</td>
<td>Timer A4 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TB0ICR-00</td>
<td>Timer B0 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TB1ICR-00</td>
<td>Timer B1 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>TB2ICR-00</td>
<td>Timer B2 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>INT0ICR-00</td>
<td>/INT0 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>INT1ICR-00</td>
<td>/INT1 intr ctrl</td>
<td></td>
</tr>
<tr>
<td>INT2ICR-00</td>
<td>/INT2 intr ctrl</td>
<td></td>
</tr>
</tbody>
</table>

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.

The Interrupt Control Registers window's 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.

### Interrupt Control Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>adcicr</td>
<td>A-D conversion intr ctrl</td>
<td>70H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>u0ticr</td>
<td>UART0 Tx intr ctrl</td>
<td>71H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>u0ricr</td>
<td>UART0 Rx intr ctrl</td>
<td>72H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ulticr</td>
<td>UART1 Tx intr ctrl</td>
<td>73H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ulricr</td>
<td>UART1 Rx intr ctrl</td>
<td>74H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta0icr</td>
<td>Timer A0 intr ctrl</td>
<td>75H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta1icr</td>
<td>Timer A1 intr ctrl</td>
<td>76H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta2icr</td>
<td>Timer A2 intr ctrl</td>
<td>77H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta3icr</td>
<td>Timer A3 intr ctrl</td>
<td>78H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta4icr</td>
<td>Timer A4 intr ctrl</td>
<td>79H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tb0icr</td>
<td>Timer B0 intr ctrl</td>
<td>7AH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tb1icr</td>
<td>Timer B1 intr ctrl</td>
<td>7BH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tb2icr</td>
<td>Timer B2 intr ctrl</td>
<td>7CH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>int0icr</td>
<td>/INT0 intr ctrl</td>
<td>7DH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>int1icr</td>
<td>/INT1 intr ctrl</td>
<td>7EH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>int2icr</td>
<td>/INT2 intr ctrl</td>
<td>7FH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
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 Return 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 Return 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.
The Mode Registers Window

The Mode Registers window displays contents of Mode Conversion Registers.

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.

The Mode Conversion Registers window’s 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.

**Mode Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmr</td>
<td>Processor mode</td>
<td>5EH</td>
<td>1</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>pmr0</td>
<td>Processor mode 0</td>
<td>5EH</td>
<td>1</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>pmr1</td>
<td>Processor mode 1</td>
<td>5FH</td>
<td>1</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
The Other Registers Window

The Other Registers window displays contents of M37734 unique Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC2=00</td>
<td>Oscillating circuit control</td>
<td></td>
</tr>
<tr>
<td>PSF=00</td>
<td>Port P5 control function contr</td>
<td></td>
</tr>
<tr>
<td>STC=00</td>
<td>Serial transmit control</td>
<td></td>
</tr>
<tr>
<td>S=00</td>
<td>Special</td>
<td></td>
</tr>
</tbody>
</table>

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.

The Other Registers window's 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.
### Other Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>occ</td>
<td>Oscillating circuit control</td>
<td>6CH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p5fc</td>
<td>Port P5 function control</td>
<td>6DH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>stc</td>
<td>Serial transmit control</td>
<td>6EH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>Special</td>
<td>6FH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
The Port Registers Window

The Port Registers window displays contents of Port Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0=00</td>
<td>Port P0</td>
<td></td>
</tr>
<tr>
<td>P1=00</td>
<td>Port P1</td>
<td></td>
</tr>
<tr>
<td>P0DDR=00</td>
<td>Port P0 data direction</td>
<td></td>
</tr>
<tr>
<td>P1DDR=00</td>
<td>Port P1 data direction</td>
<td></td>
</tr>
<tr>
<td>P2=00</td>
<td>Port P2</td>
<td></td>
</tr>
<tr>
<td>P3=00</td>
<td>Port P3</td>
<td></td>
</tr>
<tr>
<td>P2DDR=00</td>
<td>Port P2 data direction</td>
<td></td>
</tr>
<tr>
<td>P3DDR=00</td>
<td>Port P3 data direction</td>
<td></td>
</tr>
<tr>
<td>P4=03</td>
<td>Port P4</td>
<td></td>
</tr>
<tr>
<td>P5=00</td>
<td>Port P5</td>
<td></td>
</tr>
<tr>
<td>P4DDR=00</td>
<td>Port P4 data direction</td>
<td></td>
</tr>
<tr>
<td>P5DDR=00</td>
<td>Port P5 data direction</td>
<td></td>
</tr>
<tr>
<td>P6=00</td>
<td>Port P6</td>
<td></td>
</tr>
<tr>
<td>P7=00</td>
<td>Port P7</td>
<td></td>
</tr>
<tr>
<td>P6DDR=00</td>
<td>Port P6 data direction</td>
<td></td>
</tr>
<tr>
<td>P7DDR=00</td>
<td>Port P7 data direction</td>
<td></td>
</tr>
<tr>
<td>P8=00</td>
<td>Port P8</td>
<td></td>
</tr>
<tr>
<td>P8DDR=00</td>
<td>Port P8 data direction</td>
<td></td>
</tr>
</tbody>
</table>

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.

The Port Registers window’s 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.

Port Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>Port P0</td>
<td>02H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>Port P1</td>
<td>03H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p0ddr</td>
<td>Port P0</td>
<td>04H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p1ddr</td>
<td>Port P1</td>
<td>05H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>Port P2</td>
<td>06H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>Port P3</td>
<td>07H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2ddr</td>
<td>Port P2</td>
<td>08H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p3ddr</td>
<td>Port P3</td>
<td>09H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>Port P4</td>
<td>0AH</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td>Port P5</td>
<td>0BH</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4ddr</td>
<td>Port P4</td>
<td>0CH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p5ddr</td>
<td>Port P5</td>
<td>0DH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>Port P6</td>
<td>0EH</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>Port P7</td>
<td>0FH</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p6ddr</td>
<td>Port P6</td>
<td>10H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p7ddr</td>
<td>Port P7</td>
<td>11H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>Port P8</td>
<td>12H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p8ddr</td>
<td>Port P6</td>
<td>14H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
The Pulse Motor Control Registers Window

The Pulse Motor Control Registers window displays contents of Pulse Motor Control Registers.

![Pulse Motor Control Registers Window](image)

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.

The Pulse Motor Control Registers window's 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.
### Pulse Motor Control Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>womr</td>
<td>Waveform output mode</td>
<td>62H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dtt</td>
<td>Dead-time timer</td>
<td>63H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>podr1</td>
<td>Pulse output data1</td>
<td>64H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>podr0</td>
<td>Pulse output data0</td>
<td>65H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 popup menu commands.
### Debugger Windows

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>The name of the displayed source file appears at the top of the window.</td>
</tr>
<tr>
<td>Source Lines</td>
<td>C source code is displayed when available. Source lines are preceded by the corresponding line numbers.</td>
</tr>
<tr>
<td></td>
<td>When programs are written in assembly language or when no C source code is available, disassembled instruction mnemonics are displayed.</td>
</tr>
<tr>
<td>Disassembled</td>
<td>In the Mnemonic Display mode, disassembled instruction mnemonics are intermixed with the source lines.</td>
</tr>
<tr>
<td>Instructions</td>
<td>Disassembled lines contain address, data, and mnemonic information.</td>
</tr>
<tr>
<td></td>
<td>When symbolic information is available for the address, the corresponding symbol line precedes the disassembled instruction, displayed in the module_name\symbol_name format.</td>
</tr>
<tr>
<td>Current PC</td>
<td>The line associated with the current program counter is highlighted.</td>
</tr>
<tr>
<td>Scroll Bars</td>
<td>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.</td>
</tr>
<tr>
<td>&quot;BP&quot; Marker</td>
<td>The breakpoint marker, &quot;BP&quot;, appears at the beginning of the breakpoint lines or break macro lines.</td>
</tr>
<tr>
<td>Execution Coverage</td>
<td>The accessed (executed) lines are highlighted when program execution coverage is enabled.</td>
</tr>
<tr>
<td>Break Macro Lines</td>
<td>Decimal points following line numbers or addresses indicate break macro lines.</td>
</tr>
</tbody>
</table>

**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" in the "Debugging Programs" chapter, and
Making Coverage Measurements in the "Debugging Programs" chapter.

"Source Window Commands" in the "Window Control Menu Commands" chapter.
"Source Window Popup Commands" in the "Window Popup Commands" chapter.

The Status Window

The Status window shows the emulator status, the trace status, and the scope of the current program counter value.

### 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.

Up to 2000 modules, 4000 functions, or 4000 symbols can be shown in the Symbol window. When your program has more symbols than allowed, it's...
hard to predict which symbols will not be shown due to the way they're read from the object file.

See Also

"Displaying Symbol Information" in the "Debugging Programs" chapter.

"Symbol Window Commands" in the "Window Control Menu Commands" chapter.
The Timer Registers Window

The Timer Registers window displays contents of Timer Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSF=00</td>
<td>Count start flag</td>
<td></td>
</tr>
<tr>
<td>OSSF=00</td>
<td>One shot start flag</td>
<td></td>
</tr>
<tr>
<td>UDFF=00</td>
<td>Up-downflag</td>
<td></td>
</tr>
<tr>
<td>TA0=FF7F</td>
<td>Timer A0</td>
<td></td>
</tr>
<tr>
<td>TA1=FFEF</td>
<td>Timer A1</td>
<td></td>
</tr>
<tr>
<td>TA2=FFFF</td>
<td>Timer A2</td>
<td></td>
</tr>
<tr>
<td>TA3=FFFF</td>
<td>Timer A3</td>
<td></td>
</tr>
<tr>
<td>TA4=FFFF</td>
<td>Timer A4</td>
<td></td>
</tr>
<tr>
<td>TB0=FF9F</td>
<td>Timer B0</td>
<td></td>
</tr>
<tr>
<td>TB1=FFFF</td>
<td>Timer B1</td>
<td></td>
</tr>
<tr>
<td>TB2=FFFF</td>
<td>Timer B2</td>
<td></td>
</tr>
<tr>
<td>TA0MR=00</td>
<td>Timer A0 mode</td>
<td></td>
</tr>
<tr>
<td>TA1MR=00</td>
<td>Timer A1 mode</td>
<td></td>
</tr>
<tr>
<td>TA2MR=00</td>
<td>Timer A2 mode</td>
<td></td>
</tr>
<tr>
<td>TA3MR=00</td>
<td>Timer A3 mode</td>
<td></td>
</tr>
<tr>
<td>TA4MR=00</td>
<td>Timer A4 mode</td>
<td></td>
</tr>
<tr>
<td>TB0MR=20</td>
<td>Timer B0 mode</td>
<td></td>
</tr>
<tr>
<td>TB1MR=20</td>
<td>Timer B1 mode</td>
<td></td>
</tr>
<tr>
<td>TB2MR=20</td>
<td>Timer B2 mode</td>
<td></td>
</tr>
</tbody>
</table>

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.

The Timer Registers window’s 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.

**Timer Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>csf</td>
<td>Count start flag</td>
<td>40H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ossf</td>
<td>One shot start flag</td>
<td>42H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>udf</td>
<td>Up-down flag</td>
<td>44H</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer A0</td>
<td>46H</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer A1</td>
<td>48H</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer A2</td>
<td>4AH</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer A3</td>
<td>4CH</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer A4</td>
<td>4EH</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer B0</td>
<td>50H</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer B1</td>
<td>52H</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0</td>
<td>Timer B2</td>
<td>54H</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ta0mr</td>
<td>Timer A0 mode</td>
<td>56H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta1mr</td>
<td>Timer A1 mode</td>
<td>57H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta2mr</td>
<td>Timer A2 mode</td>
<td>58H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta3mr</td>
<td>Timer A3 mode</td>
<td>59H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ta4mr</td>
<td>Timer A4 mode</td>
<td>5AH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tb0mr</td>
<td>Timer B0 mode</td>
<td>5BH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tb1mr</td>
<td>Timer B1 mode</td>
<td>5CH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>tb2mr</td>
<td>Timer B2 mode</td>
<td>5DH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
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 accumulated 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→Bus Cycle ON (ALT, -, D, B) 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 UART0 Registers Window

The UART0 Registers window displays contents of UART0 Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>U0TRMR=00</td>
<td>UART0</td>
<td>Tx/Rx mode</td>
</tr>
<tr>
<td>U0BRG=00</td>
<td>UART0</td>
<td>bit rate generator</td>
</tr>
<tr>
<td>U0TR=0000</td>
<td>UART0</td>
<td>Tx buffer</td>
</tr>
<tr>
<td>U0TCR0=00</td>
<td>UART0</td>
<td>Tx/Rx ctrl 0</td>
</tr>
<tr>
<td>U0TCR1=02</td>
<td>UART0</td>
<td>Tx/Rx ctrl 1</td>
</tr>
<tr>
<td>U0RBR=43FF</td>
<td>UART0</td>
<td>Rx buffer</td>
</tr>
</tbody>
</table>

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.

The UART0 Registers window’s 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.

**UART0 Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>u0trmr</td>
<td>UART0 Tx/Rx mode</td>
<td>30H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>u0brg</td>
<td>UART0 bit rate</td>
<td>31H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>u0tbr</td>
<td>UART0 Tx buffer</td>
<td>32H</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>u0trcr0</td>
<td>UART0 Tx/Rx ctrl 0</td>
<td>34H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>u0trcr1</td>
<td>UART0 Tx/Rx ctrl 1</td>
<td>35H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>u0rbr</td>
<td>UART0 Rx buffer</td>
<td>36H</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
The UART1 Registers Window

The UART1 Registers window displays contents of UART1 Registers.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1TRMR=00</td>
<td>UART1</td>
<td>Tx/Rx mode</td>
</tr>
<tr>
<td>U1BRG=00</td>
<td>UART1</td>
<td>bit rate generator</td>
</tr>
<tr>
<td>U1TBR=0000</td>
<td>UART1</td>
<td>Tx buffer</td>
</tr>
<tr>
<td>U1TRCR0=08</td>
<td>UART1</td>
<td>Tx/Rx ctrl 0</td>
</tr>
<tr>
<td>U1TRCR1=02</td>
<td>UART1</td>
<td>Tx/Rx ctrl 1</td>
</tr>
<tr>
<td>U1RBR=03FF</td>
<td>UART1</td>
<td>Rx buffer</td>
</tr>
</tbody>
</table>

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.

The UART1 Registers window's 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.
### UART1 Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>utrnr</td>
<td>UART1 Tx/Rx mode</td>
<td>38H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ulbreg</td>
<td>UART1 bit rate</td>
<td>39H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ultbr</td>
<td>UART1 Tx buffer</td>
<td>3AH</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ultrcr0</td>
<td>UART1 Tx/Rx ctrl 0</td>
<td>3CH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ultrcr1</td>
<td>UART1 Tx/Rx ctrl 1</td>
<td>3DH</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ulrbr</td>
<td>UART1 Rx buffer</td>
<td>3EH</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Watchdog Timer Registers Window

The Watchdog Timer Registers window displays contents of Watchdog Timer Registers.

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDI=00</td>
<td></td>
<td>Watchdog timer</td>
</tr>
<tr>
<td>WDIFSF=00</td>
<td></td>
<td>Frequency selection</td>
</tr>
</tbody>
</table>
```

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.

The Watchdog Timer Registers window’s 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.
### Watchdog Timer Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Address</th>
<th>Size</th>
<th>Bit Edit</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdt</td>
<td>Watchdog Timer</td>
<td>60H</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wdtfsf</td>
<td>Frequency selection</td>
<td>61H</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

---

### 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 Return 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 ICC7700

IEEE-695 Object Files

This section addresses the IEEE-695 object files generated by following IEEE-695 converter after compiled or assembled with the following compiler and assembler:

- IAR ICC7700 Compiler
- IAR A7700 Assembler
- MRI MCCM77 Compiler
- MRI ASMM77 Assembler
- Mitsubishi NC77 Compiler
- Mitsubishi RASM77 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.

Compiling Programs with ICC7700
1 Compile the source files with the icc7700 command.
2 Assemble the source files with the a7700 command.
3 Link the object files with the xlink command.
4 Convert UBROF file to IEEE-695 file with iar2ieee command.

Required Compiler/Assembler/Linker/Converter
Compiler
    IAR ICC7700 Compiler/DOS
Assembler
    IAR A7700 Assembler/DOS
Linker
    IAR XLINK Linker/DOS
Converter
    IAR UBROF to IEEE-695 Converter

Compiling
For compiling, use the icc7700 command in your IAR C Compiler with the following option switches:

-\texttt{r} \quad \text{Generates debug information.}
-\texttt{z0} \quad \text{Optimize for size.}
-\texttt{P} \quad \text{Generates PROMable code.}
You need to use -e and -2 options when you use double float.

You can’t use -ri, -rn, and -s options to compile.

Assembling

For assembling, use the a7700 command in your IAR Assembler with the following option switch:

-\texttt{r} \quad \text{Enables debugger output.}

Since UBROF format file does not include assembler local symbol information, you can not debug assembler module using IEEE-695 file converted from UBROF file. If you need to debug assembler module, you must specify assembler local symbols by using \texttt{PUBLIC} directive.

Linking

For linking, use the xlink command in your IAR Linker with either of the following option switches:

-\texttt{r} \quad \text{Link with debug.}
-\texttt{FDEBUG} \quad \text{Link with debug.}

Converting

For converting to IEEE-695 format from UBROF format, use the iar2ieee command in your IAR converter.

See Also

"To select memory model" in the "Configuring the Emulator" chapter.
Compiling Programs with MCCM77

1 Compile the source files with the mccm77 command.

2 Assemble the source files with the asmm77 command.

3 Link the object files with the lnkm77 command.

**Required Compiler/Assembler/Linker/Converter**

- **Compiler**
  - MRI MCCM77 Compiler/DOS
- **Assembler**
  - MRI ASMM77 Assembler/DOS

**Compiling**

For compiling, use the mccm77 command in your MRI C Compiler with the following option switches:

- `-g` Generates debug information.
- `-c` Generates object file.
- `-n0g` Turn off Global-Flow optimizer.

**Assembling**

For assembling, use the asmm77 command in your MRI Assembler with the following option switch:

- `-f d` Places debug information into object file.

**Linking**

For linking, use the lnkm77 command in your MRI Linker.

**See Also**

"To select memory model" in the "Configuring the Emulator" chapter.
Compiling Programs with NC77

1 Compile the source files with the nc77 command.

2 Assemble the source files with the rasm77 command.

3 Link the object files with the link77 command.

4 Convert MELPS 7700 Hex file to IEEE-695 file with s2ie command.

**Required Compiler/Assembler/Linker/Converter**

Compiler
  Mitsubishi NC77 Compiler/DOS

Assembler
  Mitsubishi RASM77 Assembler/DOS

Linker
  Mitsubishi LINK77 Linker/DOS

Converter
  Mitsubishi s2ie Converter/DOS

**Compiling**

For compiling, use the nc77 command in your Mitsubishi C Compiler with the following option switches:

- **-gie** Generates IEEE-695 format file.

- **-c** Generates object file.

**Assembling**

For assembling, use the rasm77 command in your Mitsubishi Assembler with the following option switch:

- **-s** Generates local symbol information.

- **-c** Generates source debug information.
Chapter 13: Concepts
Compiler/Assembler Specifications

Linking

For linking, use the link77 command in your Mitsubishi Linker with the following option switch:

-s Generates local symbol information.

Converting

For converting to IEEE-695 format from MELPS Hex format, use the s2ie command.

See Also

"To select memory model" in the "Configuring the Emulator" chapter.
Monitor Programs

This section describes:

- Monitor Program Options
- Assembling, Linking and Converting the Foreground Monitor
- Notes on Foreground Monitors

The foreground monitor source file is included with the debugger software and can be found in the C:\HP\RTC\M7700\FGMON directory (if C:\HP\RTC\M7700 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.

Usually, the background monitor will be easier to work with in starting a new design. So it is recommended to use background monitor. The background monitor is immediately available and upon powerup, and you don’t worry about linking in the code or allocating space for the monitor to use the emulator.

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 files are provided with the debugger software. 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**

- You can not perform run control commands from menu bar commands, and need to create buttons to use their functions. Refer to the Notes on Foreground Monitors to create buttons.
- Step command in not available when the emulator is used with a foreground monitor.
- The foreground monitor occupies 2 Kbytes of the user memory space.
• The foreground monitor must be assembled and linked prior to use.
• You cannot perform synchronized measurements over the CMB.

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.

In the HP 64146/7 emulator, the starting address can be specified on a 2 Kbyte boundary in bank 0 other then Internal RAM area and SFR area.

To specify the foreground monitor starting address, you must modify the .EQU statement that follows the first comment in the source program as shown below.

```assembly
LOCATE_ADRS  .EQU 0xxxxH ; start monitor on 2 k byte boundary in bank 0 rather than sfr/iram area
PROCMODEREG .EQU 0xxxxH ; processor mode register’s address
```

Specify the foreground monitor address and processor mode register address by modifying 0xxxxH.

Specifying Processor Name

To use foreground monitor with 7751 microprocessor, you need to modify the processor name section of foreground monitor source. Default setting is following.

```assembly
CHIP 7751     .WORD  0      ; OTHER_THAN_7751
```

You can specify 7751 microprocessor by modifying this section like below.

```assembly
CHIP 7751     .WORD  1      ; 7751
```
Processor Mode Register Address

You may need to modify the .EQU statement at the PROCMODEREG label. This value defines the location of processor mode register. If your processor has processor mode register at address other than 5e hex, modify this value to appropriate value.
Assembling, Linking the Foreground Monitor with A7700

The foreground monitor can be assembled, linked with your Assembler and Linker.

If you want to assemble the foreground monitor with IAR A7700 assembler, enter:

C> a7700 -r fgmon.s28

To link the foreground monitor, enter:

C> xlink -r -c7700 -o fgmon.d28 fgmon.s28

To convert the foreground monitor, enter:

C> iar2ieee fgmon.d28 fgmon.x

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.

To use the HP 64146/7 emulator with the foreground monitor

When using the foreground monitor, you cannot use the run control commands from the menu bar. To run the user program, you need to create user-defined buttons and execute them instead of using the run control commands. The following user-defined buttons executes the user program from the terminal mode of the emulator.

<table>
<thead>
<tr>
<th>Button Name</th>
<th>Button Command</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERM RUN</td>
<td>termcom r</td>
<td>Run from current PC</td>
</tr>
</tbody>
</table>
If you want to execute "run from address", you need to modify Program counter to the address you want to start.

And also, if you want to do step exection, you need to set a breakpoint at address you want to stop.

To run from breakpoint address using the above-mentioned button (TERM RUN), you need to delete a breakpoint. Therefore, we recommend to create a following user-defined button that deletes all of breakpoints.

<table>
<thead>
<tr>
<th>Button Name</th>
<th>Button Command</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEL BP</td>
<td>bp delete all</td>
<td>Delete all breakpoints</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Trace Signals</th>
<th>Signal</th>
<th>Name</th>
<th>Signal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>D0-D15</td>
<td>Processor Data 0-15</td>
<td></td>
</tr>
<tr>
<td>16-39</td>
<td>A0-A23</td>
<td>Address Lines 0-23</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>R/W_L</td>
<td>0 = Write Cycle, 1 = Read Cycle</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>UDEN_L</td>
<td>0 = User Data Enable</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>DATA_L</td>
<td>0 = Data Cycle</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>EXEC_L</td>
<td>0 = Execution Cycle</td>
<td></td>
</tr>
<tr>
<td>44-45</td>
<td>MASTER0-1</td>
<td>00 = Refresh Cycle, 01 = Hold Cycle, 10 = DMA Cycle, 11 = CPU Cycle</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>BKG</td>
<td>0 = Foreground Cycle, 1 = Background Cycle</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>MX</td>
<td>1 = MX Cycle</td>
<td></td>
</tr>
</tbody>
</table>
### Predefined Status Values

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Status Bits (47-40)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bg</td>
<td>0x1xx xxxxxy</td>
<td>Background cycle</td>
</tr>
<tr>
<td>byte</td>
<td>0xx1x 1x1xy</td>
<td>Byte access cycle</td>
</tr>
<tr>
<td>cpu</td>
<td>0xx11 xxxxxy</td>
<td>CPU cycle</td>
</tr>
<tr>
<td>data</td>
<td>0xx1x 10xxyn</td>
<td>Data access cycle</td>
</tr>
<tr>
<td>dataread</td>
<td>0xx1x 10x1yn</td>
<td>Data read cycle</td>
</tr>
<tr>
<td>datawrite</td>
<td>0xx1x 10x0yn</td>
<td>Data access cycle</td>
</tr>
<tr>
<td>dma</td>
<td>0xx10 xxxxxy</td>
<td>DMA cycle</td>
</tr>
<tr>
<td>dmaread</td>
<td>0xx10 1xx1yn</td>
<td>DMA read cycle</td>
</tr>
<tr>
<td>dmawrite</td>
<td>0xx10 1xx0yn</td>
<td>DMA write cycle</td>
</tr>
<tr>
<td>exec</td>
<td>0xx11 01xxyn</td>
<td>Execution cycle</td>
</tr>
<tr>
<td>fetch</td>
<td>0xx11 11x1yn</td>
<td>Fetch cycle</td>
</tr>
<tr>
<td>fg</td>
<td>0x0xx xxxxxy</td>
<td>Foreground cycle</td>
</tr>
<tr>
<td>hold</td>
<td>0xx01 xxxxxy</td>
<td>Hold cycle</td>
</tr>
<tr>
<td>mx</td>
<td>01xxx xxxxxy</td>
<td>MX cycle</td>
</tr>
<tr>
<td>read</td>
<td>0xx1x 1xx1yn</td>
<td>Read cycle</td>
</tr>
<tr>
<td>ref</td>
<td>0xx00 xxxxxy</td>
<td>Refresh cycle</td>
</tr>
<tr>
<td>word</td>
<td>0xx1x 1x0xyn</td>
<td>Word access cycle</td>
</tr>
<tr>
<td>write</td>
<td>0xx1x 1xx0yn</td>
<td>Write cycle</td>
</tr>
</tbody>
</table>
Part 5

Installation Guide

Instructions for installing the product.
Part 5
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 80386 microprocessor and 4 megabytes of memory.
- MS Windows 3.1.
- 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.

As with many Windows applications, 8 megabytes or more of memory and an 80486 microprocessor will measurably improve performance.
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.

- 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 DIP 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.

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 light at the lower right-hand corner of the front panel will be illuminated.
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, Xon/Xoff Flow Control, and the PC's RS-232 interface connector. Choose the OK button.

You should now be able to press the Return key in the Terminal window to see the HP 64700's Terminal Interface prompt (for example, "R>", "M>", "U>", etc.). 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 a 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
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 may or may not be shown, depending on the HP 64700 model. 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" lines 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 Return 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 DIP switches on the HP 64700 rear panel, and set them as shown below.

![DIP switch configuration](image)

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 light at the lower right-hand corner of the front panel will be illuminated.
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 B3630.INI file because it will be overwritten by the installation process.

2. Start MS Windows in the 386 enhanced mode.

3. Insert the Real-Time C Debugger system disk 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.
Chapter 14: Installing the Debugger

If you cannot verify LAN communication

You will be asked to enter the installation path. The default installation path is C:\HP\RTC\M7700. The default installation path is shown wherever files are discussed in this manual.

![Installation Path for HP Real-Time C](image)

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.

![User Identification](image)
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.

After you have specified the type of connection, files will be copied to your hard disk. (The B3630.TMP and B3630.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 re-install 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 "M7700 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: Xon/Xoff
  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:

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9600</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>19200</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2400</td>
</tr>
</tbody>
</table>

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 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.

With certain RS-232 cards, connecting to an RS-232 port where the HP 64700 is turned OFF (or is not connected) will hang the PC. The only way to get control back is to reboot the PC. Therefore, we recommend that you always turn ON the HP 64700 before attempting to connect via RS-232.
If you have LAN connection problems

☐ Try to "ping" the emulator:

```
ping <hostname or IP address>
```

If the emulator does not respond:

1. Check that switch 16 on the emulator is "1" (emulator is attached to LAN, not RS-232 or RS-422).

2. 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 power cycle the emulator.)

☐ 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 subnet 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. 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".

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, turn power back ON, and hope it doesn't happen again. Also,
you could place 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:

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>230400</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>115200</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>38400</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>57600</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1200</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2400</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>19200</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9600</td>
</tr>
</tbody>
</table>

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 80386 microprocessor and 4 megabytes of memory. Acceptable performance is usually found on 25 Mhz 386DX systems and faster. 386SX and slower speed systems will probably feel quite sluggish.

- 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.
Installing/Updating HP 64700 Firmware
Installing/Updating HP 64700 Firmware

This chapter shows you how to install or update HP 64700 firmware.

**Note**

Your HP 64700 must contain Flash EPROM memory before you can install or update HP 64700 system firmware.

The firmware, and the program that downloads it into the HP 64700, are included with the debugger on floppy disks labeled HP 64700 EMULATION AND ANALYSIS FIRMWARE.

The steps to install or update HP 64700 firmware are:

- Step 1. Connect the HP 64700 to the PC
- Step 2. Install the firmware update utility
- Step 3. Run PROGFLASH to update HP 64700 firmware
Step 1. Connect the HP 64700 to the PC

1 Set the HP 64700 configuration switches for RS-232C communication. Locate the DIP switches on the HP 64700 rear panel, and set them as shown below.

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>
```

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.

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 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 light at the lower right-hand corner of the front panel will be illuminated.
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: 9600 Baud Rate, 8 Data Bits, 1 Stop Bit, Parity None, Xon/Xoff Flow Control, and the PC's RS-232 interface connector to which the RS-232 cable was attached. Choose the OK button.

You should now be able to press the Return key in the Terminal window to see the HP 64700's Terminal Interface prompt (for example, "R>", "M>", "U>", etc.). 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".

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 EMULATION AND ANALYSIS FIRMWARE 1 of 2 disk 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.
Follow the remaining instructions to install the firmware update utility and the HP 64700 system firmware.

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 AUTOEXEC.BAT file to set the HP64700 and HPTABLES environment variables. For example:

```
SET HP64700=C:\INSTPATH
SET HPTABLES=C:\INSTPATH\TABLES
```

6 If you are using the COM3 or COM4 ports, you need to edit the <installation_path>\TABLES\64700TAB file. The default 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
```

Either add another line or modify one of the existing lines. For example:

```
EMUL_COM3 unknown COM3 OFF 9600 NONE ON 1 8
EMUL_COM4 unknown COM4 OFF 9600 NONE ON 1 8
```

Firmware update utility installation is now complete. The PC needs to be rebooted to enable the changes made to the CONFIG.SYS and AUTOEXEC.BAT files. To reboot, press the CTRL+ALT+DEL keys simultaneously.
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 (in other words, HP 64700) you want to update.

5 Enter the number that identifies the product whose firmware you want to update.

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.

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.
**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.

**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.

**coverage measurements**  Measurements which show the code that has been accessed during program execution.

**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.

**mx flag**  The m flag determines whether the data length is 16-bit or 8-bit. The data length is 16-bit when flag m is "0" and 8-bit when it is "1".

The x flag determines whether index register X and index register Y are used as 16-bit registers or as 8-bit registers. The registers are used as 16-bit registers when flag x is "0" and as 8-bit registers when it is "1".

**object file**  An IEEE-695 format absolute file that can be loaded into emulation or target system memory and executed by the debugger.

**popup 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 M37700 emulator are:

- **bg** Background cycle
- **byte** Byte access cycle
- **cpu** CPU cycle
- **data** Data access cycle
- **dataread** Data read cycle
- **datawrite** Data write cycle
- **dma** DMA cycle
- **dmaread** DMA read cycle
- **dmawrite** DMA write cycle
- **exec** Execution cycle
- **fetch** Fetch cycle
- **fg** Foreground cycle
- **hold** Hold cycle
- **mx** MX flag cycle
- **read** Read cycle
- **ref** Refresh cycle
- **word** Word access cycle
- **write** Write cycle
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.
Glossary Defines terms that are used in the debugger help information.
Index

! /RDY signal, enabling or disabling, 79
A A-D Conversion Registers window, 402, 406
A7700, assembling and linking the foreground monitor with, 447
accumulated count information, displaying, 159, 363
Add to Watch command, 375
addresses, searching, 108, 349
analyzer, 485-487, 489-490
analyzer trace signals, 449-450
analyzer, editing the trace specification, 173, 244
analyzer, halting, 157, 257
analyzer, repeating last trace, 157, 258
analyzer, setting up with "Find Then Trigger", 164, 248-251
analyzer, setting up with "Sequence", 168, 252-255
analyzer, setting up with "Trigger Store", 161, 245-247
analyzer, tracing until halt, 157, 256
appendix "@i", 135
arguments, function, 404, 438, 440-441
arm condition, 94, 164, 168, 259-261, 295, 485-487, 489-490
arrays (C operators), 192
ASCII values in Memory window, 133, 412
Assemble... (ALT, A) command, 275
assembler, in-line, 275
assembly instructions, specifying mx flag, 105
assembly language instructions, stepping multiple, 120, 220-222
assembly language instructions, stepping single, 118, 218
assembly language source files, 437
auto variables, 130-132
AUTOEXEC.BAT file, 481-482
B background memory, 485-487, 489-490
background monitor, 443
background monitor program, 485-487, 489-490
background monitor, selecting, 88, 283-285
background operation, tracing, 383
BackTrace window, 404
Index

BackTrace window, displaying source files, 373
Basic Registers window, 405
beep, sounding from command file, 379
BNC port, driving the trigger signal, 293-294
BNC port, output the trigger signal, 94
BNC port, receiving an arm condition from, 295
BNC port, receiving an arm condition input from, 94
BNC port, setting up, 94
BP marker, 33, 35, 127, 225-230, 421
break into monitor, 122, 223
break macro, 485-487, 489-490
break macros, command summary, 180
break macros, deleting, 129, 230
break macros, listing, 127, 231-232
break macros, setting, 127, 227-229
break on writes to ROM, enabling or disabling, 80
breakpoint, 485-487, 489-490
Breakpoint-Delete at Cursor (ALT, B, D) command, 226
Breakpoint-Delete Macro (ALT, B, L) command, 230
Breakpoint-Edit... (ALT, B, E) command, 231-232
Breakpoint-Set at Cursor (ALT, B, S) command, 225
Breakpoint-Set Macro... (ALT, B, M) command, 227-229
breakpoints, deleting, 35, 126, 226
breakpoints, disabling, 126
breakpoints, listing, 127, 231-232
breakpoints, setting, 33, 125, 225
bus cycles, displaying, 158, 362
Button window, 406
Button window, editing, 65, 321
buttons that execute command files, creating, 65

C
C operators, 192
callers (of a function), tracing, 47, 151, 237-238
Clear Breakpoint command, 374
clipboard, 55
clock source (emulator), selecting, 75, 82
clock speeds greater than 16 MHz, 81
command file execution, exiting, 381
command files, command summary, 180
command files, comments, 384
command files, creating, 63, 200
command files, executing, 64, 203-204
command files, inserting wait delays, 385-386
command files, locating cursor, 380
command files, parameters, 203-204
command files, sounding beep, 379
command files, turning logging on or off, 201-202
command line options, 58
command summary, 180
comments in command files, 384
communications (emulator), setting up, 290-292
CONFIG.SYS file, 481-482
configuration, emulator, 276-279
configurations, saving and loading, 95-96
Continuous Update (ALT, -, U) command, 338
control menu, 485-487, 489-490
Copy-Destination... (ALT, -, P, D) command, 320
Copy-Registers (ALT, -, P, R) command, 338
Copy-Window (ALT, -, P, W) command, 319
count condition, 485-487, 489-490
count conditions, 259-261
count information, displaying accumulated, 159, 363
count information, displaying relative, 159, 364
coverage (execution), displaying, 146, 296-297
coverage (execution), resetting, 298
coverage measurements, 485-487, 489-490
CTRL key and double-clicks, 55
cursor, locating from command file, 380
cut and paste, 55

D
DCE or DTE selection and RS-232 cable, 465
detector overview, 4
detector startup options, 58
detector windows, opening, 60
detector, arranging icons in window, 309
detector, cascaded windows, 309
detector, exiting, 49-50, 58, 210
detector, exiting locked, 211
detector, installing software, 467
detector, opening windows, 310-312
detector, starting, 26, 57, 470
detector, tiled windows, 309
demo program, loading, 31
demo program, mapping memory, 29-30
Index

demo program, running, 34
demo programs, 24
DeMorgan’s law, 259-261
dialog boxes, file selection, 212
directories, search path, 350
directories, source, 314
display internal RAM and SFR, 135
display mode, mixed, 104-105
display mode, source only, 104
display mode, toggling, 341-342
Display-Select Source... (ALT, -, D, L) command, 343
DMA cycles, trace, 91
do while statements (C), single-stepping, 437
double-clicks and the CTRL key, 55
dynamic variables, 233-234, 368, 436
emulation memory, 485-487, 489-490
emulation memory model, selecting, 82
evaluation memory, copying target system memory into, 139, 333
emulation microprocessor, resetting, 123, 224
emulation monitor, 485-487, 489-490
emulation monitor programs, 443
emulation pod, pod configuration, 93
emulation RAM, mapping internal RAM and SFR, 86
emulator, 485-487, 489-490
emulator clock source, selecting, 75
emulator configuration, 74, 276-279
emulator configuration, loading, 96, 207
emulator configuration, saving, 95, 208
emulator hardware options, setting, 75-82
enable condition, 485-487, 489-490
enable store condition, 485-487, 489-490
environment variables, 107
environment variables, HP64700, 481-482
environment variables, HPTABLES, 481-482
environment variables, PATH, 481-482
environment, loading, 205
environment, saving, 206
ethernet address, 460-463
Evaluate It command, 374
execution coverage, displaying, 146, 296-297
execution coverage, resetting, 298
Execution-Break (F4), (ALT, E, B) command, 223
Execution-Reset (ALT, E, E) command, 224
Execution-Run (F5), (ALT, E, U) command, 213
Execution-Run to Caller (ALT, E, T) command, 215
Execution-Run to Cursor (ALT, E, C) command, 214
Execution-Run... (ALT, E, R) command, 216-217
Execution-Single Step (F2), (ALT, E, N) command, 218
Execution-Step Over (F3), (ALT, E, O) command, 219
Execution-Step... (ALT, E, S) command, 220-222
exiting command file execution, 381
Expression window, 408
Expression window, clearing, 323
Expression window, displaying expressions, 324
expressions, 188
expressions, displaying, 324
extended settings, 302-308
externals, displaying symbol information, 112, 352
file selection dialog boxes, 212
File-Command Log-Log File Name... (ALT, F, C, N) command, 200
File-Command Log-Logging OFF (ALT, F, C, F) command, 202
File-Command Log-Logging ON (ALT, F, C, O) command, 201
File-Copy Destination... (ALT, F, P) command, 209
File-Exit (ALT, F, X) command, 210
File-Exit HW Locked (ALT, F, H) command, 211
File-Load Debug... (ALT, F, D) command, 205
File-Load Emulator Config... (ALT, F, E) command, 207
File-Load Object... (ALT, F, L) command, 197-199
File-Run Cmd File... (ALT, F, R) command, 203-204
File-Save Debug... (ALT, F, S) command, 206
File-Save Emulator Config... (ALT, F, V) command, 208
firmware update utility, installing, 481-482
firmware update, connecting the HP 64700 to the PC, 479-480
firmware version information, 313
firmware, using PROGFLASH to update, 483
font settings, 299-300
fonts, changing, 61
for statements (C), single-stepping, 437
foreground memory, 485-487, 489-490
foreground monitor, 443
foreground monitor program, 485-487, 489-490
foreground monitor, assembling, linking with A7700, 447
foreground monitor, notes on, 447
foreground monitor, selecting, 89, 283-285
foreground operation, tracing, 383
function arguments, 404, 438, 440-441
function codes, 485-487, 489-490
function keys, 56
functions, displaying symbol information, 111, 352
functions, running until return, 41, 121, 215
functions, searching, 108, 347
functions, stepping over, 42, 119, 219
functions, tracing callers, 47, 151, 237-238
functions, tracing execution within, 153, 239
functions, tracing flow, 46, 150, 236

G

gateway address, 460-463
global assembler symbols, displaying, 114, 354
global symbols, displaying, 112, 352
global variables, 112, 154-155, 352
glossary, 485-487, 489-490
guarded memory, 84, 280-282, 423, 485-487, 489-490

H

hardware options, setting, 75-82
hardware requirements, 455
hardware, locking on exit, 211
Help-About Debugger/Emulator... (ALT, H, D) command, 313
high speed access, 78
high speed access, enabling or disabling, 78
Hold cycles, trace, 92
hostname, 290-292
HP 64146/7 emulator, plugging-in, 69-70
HP 64700 firmware update utility, installing, 481-482
HP 64700 firmware update, connecting the HP 64700 to the PC, 479-480
HP 64700 firmware, using PROGFLASH to update, 483
HP 64700, connecting to the PC, 457
HP 64700, connecting via LAN, 460-463
HP 64700, connecting via RS-232, 457-459
HP 64700, connecting via RS-422, 464
HP64700 environment variable, 481-482
HPTABLES environment variable, 481-482
I/O locations, displaying, 142
I/O locations, editing, 143
I/O locations, guarding, 269-270
I/O locations, specifying, 325
I/O window, 411
I/O window, turning polling ON or OFF, 98
ICC7700, compiling programs with, 438
icon, for a different emulator, 58
icons (debugger window), arranging, 309
IEEE-695 object files, 437
in-circuit operation, configuring the emulator for, 71-72
in-line assembler, 275
installation path, 467
internal RAM and SFR, displaying, 135
internal RAM and SFR, mapping, 86
internal RAM/ROM/SFR, in pod configuration, 93, 288-289
internals, displaying symbol information, 113, 353
Internet Address, 290-292, 460-463, 466-469
Interrupt Control Registers window, 409
interrupts (target system), 443
interset operators, 259-261
intraset operators, 259-261
intricate RDY signal, 79
intrusion, monitor, 98, 267-268
labels, 190-191, 275
LAN cards, 455-456
LAN communication, 290-292, 470
LAN, connecting HP 64700, 460-463
levels, trace sequence, 168, 173, 252-255, 266
limitations, Symbol window, 426
line (source file), running until, 43, 121, 214
link level address, 460-463
list file, changing the destination, 61
list file, copying window contents to, 60
listing files, specifying, 209, 320
local assembler symbols, displaying, 114, 355
local symbols, displaying, 113, 353
local variables, 113-114, 353
lock hardware on exit, 211
log (command) files, 63, 200-204
logical operators, 164, 168, 259-261
M  macro, 485-487, 489-490
    MCCM77, compiling programs with, 440
    memory (target system), copying into emulation memory, 139, 333
    memory type, 84, 280-282
    Memory window, 412
    Memory window, displaying 16-bit values, 328
    Memory window, displaying 32-bit values, 329
    Memory window, displaying bytes, 328
    Memory window, displaying multi-column format, 328
    Memory window, displaying single-column format, 327
    Memory window, turning polling ON or OFF, 98
    memory, copying, 138, 331
    memory, displaying, 133
    memory, editing, 137
    memory, loading from stored file, 335
    memory, mapping, 83-87, 280-282
    memory, mapping for demo program, 29-30
    memory, modifying a range, 140, 332
    memory, searching for a value or string in, 141
    memory, storing to a binary file, 336
    microprocessor, resetting, 123, 224
    mixed display mode, 104, 341, 437
    Mode Registers window, 413
    monitor, 485-487, 489-490
    monitor intrusion, 98, 267-268
    monitor programs, 443
    monitor, selecting the type, 88-90
    mx flag, display mixed mode, 105

N  NC77, compiling programs with, 441
    no-operation command, 384
    numeric constants, 189

O  object file, 485-487, 489-490
    object files, IEEE-695, 437
    object files, loading, 103, 197-199
    object files, loading the foreground monitor, 89
    operators, C, 192
    operators, interset, 259-261
    operators, intraset, 259-261
    operators, logical, 164, 168, 259-261
    optimization option, compiler, 146, 438, 440-441
options, command line, 58
Other Registers window, 415
overview, 4

P
parameters, command file, 203-204
paste, cut and, 55
PATH environment variable, 481-482
path for source file search, 107, 350
paths for source files, prompting, 382
patterns, trace, 164, 168, 248-255, 259-263
PC, connecting HP 64700, 457
performance (PC), optimizing for the debugger, 476
platform requirements, 455
pointers (C operators), 192
polling for debugger windows, turning ON or OFF, 98
popup menu, 485-487, 489-490
popup menus, accessing, 372
Port Registers window, 417
port, BNC, 94, 259-261, 293-295
port, communication, 290-292
pragma statements (C), source file information, 437
primary branch condition, 168, 252-255, 485-487, 489-490
processor mode, selecting, 78
processor mode, setting for demo program, 27
processor type, selecting, 77
processor, resetting, 123, 224
PROGFLASH firmware update utility, 483
program counter, 118, 122, 213, 216-217, 220-222, 421
program modules, displaying symbol information, 111, 351
programs, compiling with ICC7700, 438
programs, compiling with MCCM77, 440
programs, compiling with NC77, 441
programs, demo, 24
programs, loading, 103, 197-199
programs, running, 122, 213, 216-217
programs, stopping execution, 122
Pulse Motor Control Registers window, 419

Q qualifier, state, 161, 245-247
Index

R  real-time mode, disabling, 98, 268
real-time mode, enabling, 98, 267
real-time options, setting, 97-100
RealTime-I/O Polling-OFF (ALT, R, I, F) command, 270
RealTime-I/O Polling-ON (ALT, R, I, O) command, 269
RealTime-Memory Polling-OFF (ALT, R, M, F) command, 274
RealTime-Memory Polling-ON (ALT, R, M, O) command, 273
RealTime-Monitor Intrusion-Allowed (ALT, R, T, A) command, 268
RealTime-Monitor Intrusion-Disallowed (ALT, R, T, D) command, 267
RealTime-Watchpoint Polling-OFF (ALT, R, W, F) command, 272
RealTime-Watchpoint Polling-ON (ALT, R, W, O) command, 271
Refresh cycles, trace, 92
register variables, 438, 440-441
Register window, copying information from, 338
Register windows, continuous update, 338
registers, displaying, 44-45, 144
registers, editing, 145
relative count information, displaying, 159, 364
requirements, hardware, 455
requirements, platform, 455
reset, coverage, 146, 298
reset, emulator, 123, 224
reset, emulator status, 423
reset, running from target system, 122, 216-217
reset value for stack, 28
restart condition, 164, 248-251, 485-487, 489-490
return (function), running until, 41, 121, 215
ROM, enabling or disabling breaks on writes to, 80
RS-232 cable and DCE or DTE selection, 465
RS-232, connecting HP 64700, 457-459
RS-422, connecting HP 64700, 464
run to caller, interrupting, 391
Run to Cursor command, 375
S  search path for source files, 107, 350
Search-Address... (ALT, -, R, A) command, 349
Search-Function... (ALT, -, R, F) command, 347
Search-String... (ALT, -, R, S) command, 346
Search... (ALT, -, R) command, 329
secondary branch condition, 168, 252-255, 485-487, 489-490
sequence levels, 266, 485-487, 489-490
service ports, TCP, 460-463
Set Breakpoint command, 374
Settings-BNC-BNC Drive Trigger (ALT, S, B, D) command, 293-294
Settings-BNC-BNC Receive Arm (ALT, S, B, R) command, 295
Settings-Communication... (ALT, S, C) command, 290-292
Settings-Coverage-Coverage OFF (ALT, S, V, F) command, 297
Settings-Coverage-Coverage ON (ALT, S, V, O) command, 296
Settings-Coverage-Coverage Reset (ALT, S, V, R) command, 298
Settings-Emulator Config-Hardware... (ALT, S, E, H) command, 276-279
Settings-Emulator Config-Memory Map... (ALT, S, E, M) command, 280-282
Settings-Emulator Config-Monitor... (ALT, S, E, O) command, 283-285
Settings-Emulator Config-Pod... (ALT, S, E, P) command, 288-289
Settings-Emulator Config-Trace Options... (ALT, S, E, T) command, 286-287
Settings-Extended-Load Error Abort-On (ALT, S, X, L, O) command, 305-306
Settings-Extended-Source Path Query-On (ALT, S, X, S, O) command, 307-308
Settings-Extended-Trace Cycles-Both (ALT, S, X, T, B) command, 304
Settings-Extended-Trace Cycles-Monitor (ALT, S, X, T, M) command, 303
Settings-Extended-Trace Cycles-User (ALT, S, X, T, U) command, 302
Settings-Font... (ALT, S, F) command, 299-300
Settings-Tabstops... (ALT, S, T) command, 301
single-step one line, 36
software, installing debugger, 467
Source at Stack Level command, 373
source directory, 314
source display mode, toggling, 341-342
source file line, running until, 43, 121, 214
source files, displaying, 32, 106, 343
source files, displaying from BackTrace window, 373
source files, information generated for pragma statements, 437
source files, prompting for paths, 382
source files, searching for addresses, 108, 349
source files, searching for function names, 108, 347
source files, searching for strings, 109, 346
source files, specifying search directories, 107
source lines, stepping multiple, 120, 220-222
source lines, stepping single, 118, 218
source only, displaying, 104, 363
source only, displaying in Memory window, 341-342
Source window, 421
Source window, setting tabstops, 62
Source window, specifying mx flag, 344-345
Source window, toggling the display mode, 341-342
SRCPATH environment variable, 107
stack pointer, setting up the reset value, 28
startup options, 58
state qualifier, 161, 245-247, 485-487, 489-490
status register, editing, 339
status values, 449-450, 485-487, 489-490
Status window, 423
step multiple lines, 37
step one line, 36
step over, interrupting, 391
store condition, 485-487, 489-490
store conditions, 259-261
strings, displaying symbols containing, 117, 358
strings, searching memory for, 141, 329
strings, searching source files, 109, 346
structures (C operators), 192
subnet mask, 460-463
subroutines, stepping over, 219
Symbol window, 426
Symbol window, copying information, 357-358
Symbol window, searching for strings, 358
symbols, 190-191
system setup, 456

T

tabstop settings, 301
tabstops in the Source window, setting, 62
target system, 485-487, 489-490
target system interrupts, 443
target system memory, copying into emulation memory, 139, 333
TCP service ports, 460-463
telnet, 460-463, 466-469
text, selecting, 55
Timer Registers window, 428
tool type, 81
trace DMA cycles, 286-287
trace DMA cycles, trace option, 91
trace foreground/background operation, 383
trace Hold cycles, 286-287
trace Hold cycles, trace option, 92
trace options, 286-287
trace patterns, 164, 168, 248-255, 259-263
trace range, 264-265
trace Refresh cycles, 286-287
trace Refresh cycles, trace option, 92
trace settings, 259-261
trace signals, 449-450
trace specification, copying, 366
trace specification, editing, 173, 244
trace specification, loading, 176
trace specification, specifying the destination, 367
trace specification, storing, 175
trace state, 485-487, 489-490
trace state, searching for in Trace Window, 366
Trace window, 430
Trace window, copying information, 364-365
Trace window, displaying accumulated count information, 363
Trace window, displaying bus cycles, 362
Trace window, displaying relative count information, 364
Trace window, displaying source only, 363
trace, trace options, 91-92
Trace-Again (F7), (ALT, T, A) command, 258
Trace-Edit... (ALT, T, E) command, 244
Trace-Find Then Trigger... (ALT, T, D) command, 248-251
Trace-Function Caller... (ALT, T, C) command, 237-238
Trace-Function Flow (ALT, T, F) command, 236
Trace-Function Statement... (ALT, T, S) command, 239
Trace-Halt (ALT, T, H) command, 257
Trace-Sequence... (ALT, T, Q) command, 252-255
Trace-Trigger Store... (ALT, T, T) command, 245-247
Trace-Until Halt (ALT, T, U) command, 256
Trace-Variable Access... (ALT, T, V) command, 240-241
Trace-Variable Break... (ALT, T, B) command, 242-243
transfer address, 34, 120, 122, 216-217, 220-222, 485-487, 489-490
trigger, 485-487, 489-490
trigger condition, 485-487, 489-490
trigger position, 485-487, 489-490
trigger state, searching for in Trace window, 365
trigger store condition, 485-487, 489-490
tutorial, 24
type of memory, 84, 280-282

UART0 Registers window, 431
UART1 Registers window, 433
unary minus operator, 192
unions (C operators), 192
user ID, 467
user programs, loading, 103
user-defined symbols, creating, 115, 359
user-defined symbols, deleting, 117, 361
user-defined symbols, displaying, 116, 357
Utilities-Copy... (ALT, -, U, C) command, 331
Utilities-Fill... (ALT, -, U, F) command, 332
Utilities-Image... (ALT, -, U, I) command, 333
Utilities-Load... (ALT, -, U, L) command, 335
Utilities-Store... (ALT, -, U, S) command, 336
V
values, searching memory for, 141, 329
Variable-Edit... (ALT, V, E) command, 233-234
variables, auto, 130-132
variables, displaying, 38, 130
variables, dynamic, 233-234, 368, 436
variables, editing, 39, 131, 233-235
variables, environment, 107
variables, global, 112, 154-155, 352
variables, local, 113-114, 353
variables, monitoring in the WatchPoint window, 40, 132
variables, register, 438, 440-441
variables, tracing a particular value and breaking, 155, 242-243
variables, tracing accesses, 48, 154, 240-241
version information, 313, 475
W
WAIT command, 315-316
wait delays, inserting in command files, 385-386
watchdog timer, 80
Watchdog Timer Registers window, 435
Watchdog Timer setting, enabling or disabling, 80
watchpoint, 485-487, 489-490
WatchPoint window, 436
WatchPoint window, monitoring variables in, 40, 132
WatchPoint window, turning polling ON or OFF, 98
watchpoints, editing, 368
while statements (C), single-stepping, 437
window contents, copying to the list file, 60
Window-1-9 (ALT, W, 1-9) command, 310-311
Window-Arrange Icons (ALT, W, A) command, 309
Window-Cascade (ALT, W, C) command, 309
Window-More Windows... (ALT, W, M) command, 312
Window-Tile (ALT, W, T) command, 309
windows (debugger), opening, 310-312
windows of program execution, tracing, 173
write cycles, to emulation RAM, internal RAM and SFR, 86
writes to ROM, enabling or disabling breaks on, 80
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.

Frame or chassis terminal. A connection to the frame (chassis) of the equipment 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.