User's Guide

Real-Time C Debugger for 80386

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.

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

Printing History

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

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

Edition 1 B3637-97000, November 1994

Safety, Certification and Warranty

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

Real-Time C Debugger — Overview

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

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

The debugger is an MS Windows application

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

The debugger communicates at high speeds

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

You can debug programs in C context

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

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

You can display and modify processor resources

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

You can trace program execution

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

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

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

In This Book

This book documents the Real-Time C Debugger for i80386. It is organized into five parts whose chapters are described below.

Part 1. Quick Start Guide

Chapter 1 quickly shows you how to use the debugger.

Part 2. User's Guide

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

Part 3. Reference

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

Part 4. Concept Guide

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

Part 5. Installation Guide

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

Contents

Part 1 Quick Start Guide

1 Getting Started

Step 1. Start the debugger 5
Step 2. Adjust the fonts and window size 6
Step 3. Map memory for the demo program 7
Step 4. Set address translations for the demo program 8
Step 5. Load the demo program 9
Step 6. Display the source file 10
Step 7. Set a breakpoint 11
Step 8. Run the demo program 12
Step 9. Delete the breakpoint 13
Step 10. Single-step one line 13
Step 11. Single-step 10 lines 14
Step 12. Display a variable 15
Step 13. Edit a variable 16
Step 14. Monitor a variable in the WatchPoint window 17
Step 15. Run until return from current function 18
Step 16. Step over a function 18
Step 17. Run the program to a specified line 19
Step 18. Display register contents 20
Step 19. Trace a function's callers 22
Step 20. Trace access to a variable 23
Step 21. Exit the debugger 24

Part 2 User's Guide

2 Using the Debugger Interface

Using the Debugger Interface 28

How the Debugger Uses the Clipboard 28
Debugger Function Key Definitions 29

Starting and Exiting the Debugger 30

To start the debugger 30
To exit the debugger 31
To create an icon for a different emulator 32

Working with Debugger Windows 33

To open debugger windows 33
To copy window contents to the list file 34
To change the list file destination 34
To change the debugger window fonts 35
To set tab stops in the Source window 35

Using Command Files 36

To create a command file 36
To execute a command file 37
To create buttons that execute command files 38

3 Plugging the Emulator into Target Systems

Plugging the Emulator into Target Systems 40

Step 1. Turn OFF power 41
Step 2. Unplug the probe from the demo target system 41
Step 3. Plug the probe into the target system 42
Step 4. Connect the reset flying lead to the target system 43
Step 5. Turn ON power 44

4 Configuring the Emulator

Configuring the Emulator 46

Setting the Hardware Options 47
To specify a CLK2 speed faster than 60 MHz 48
To enable or disable target interrupts 49
To enable or disable software breakpoints 50
To enable or disable break on writes to ROM 51
To enable or disable execution trace messages 52
To enable or disable foreground monitor traced as user 53

Selecting the Type of Monitor 54

To select the background monitor 54
To select the foreground monitor 55
To use a custom foreground monitor 56

Mapping Memory 58

To map memory 60

Selecting Address Translations 62

Setting Up the BNC Port 64

To output the trigger signal on the BNC port 64
To receive an arm condition input on the BNC port 64

Saving and Loading Configurations 65

To save the current emulator configuration 65
To load an emulator configuration 66

Setting the Real-Time Options 67

To allow or deny monitor intrusion 68
To turn polling ON or OFF 69

5 Debugging Programs

Debugging Programs 72

Loading and Displaying Programs 73

To load user programs 73
To display source code only 74

To display source code mixed with assembly instructions 74
To display source files by their names 75
To specify source file directories 76
To search for function names in the source files 77
To search for addresses in the source files 77
To search for strings in the source files 78

Displaying Symbol Information 79

To display program module information 80
To display function information 80
To display external symbol information 81
To display local symbol information 82
To display global assembler symbol information 83
To display local assembler symbol information 83
To create a user-defined symbol 84
To display user-defined symbol information 85
To delete a user-defined symbol 85
To display the symbols containing the specified string 86

Stepping, Running, and Stopping the Program 87

To step a single line or instruction 87
To step over a function 88
To step multiple lines or instructions 89
To run the program until the specified line 90
To run the program until the current function return 90
To run the program from a specified address 91
To stop program execution 91
To reset the processor 92

Using Breakpoints and Break Macros 93

To set a breakpoint 94
To disable a breakpoint 95
To delete a single breakpoint 95
To list the breakpoints and break macros 96
To set a break macro 96
To delete a single break macro 98

Displaying and Editing Variables 99

To display a variable 99
To edit a variable 100

To monitor a variable in the WatchPoint window 101

Displaying and Editing Memory 102

To display memory 102
To edit memory 104
To copy memory to a different location 105
To copy target system memory into emulation memory 106
To modify a range of memory with a value 107
To search memory for a value or string 108

Displaying and Editing GDT, LDT, and IDT Windows 109

To display the GDT, LDT, and IDT windows 109
To edit the GDT, LDT, and IDT windows 110

Displaying and Editing I/O Locations 111

To display I/O locations 111
To edit an I/O location 112

Displaying and Editing Registers 113

To display registers 113
To edit registers 115

Tracing Program Execution 116

To trace callers of a specified function 119
To trace execution within a specified function 121
To trace accesses to a specified variable 122
To trace until the command is halted 123
To stop a running trace 123
To repeat the last trace 123
To display bus cycles 124
To display accumulated or relative counts 125

Setting Up Custom Trace Specifications 126

To set up a "Trigger Store" trace specification 127
To set up a "Find Then Trigger" trace specification 130
To set up a "Sequence" trace specification 134
To edit a trace specification 139
To trace "windows" of program execution 139
To store the current trace specification 141
To load a stored trace specification 142

Part 3 Reference

6 Command File and Macro Command Summary

Command File and Macro Command Summary 146
WAIT Command Dialog Box 152

7 Expressions in Commands

Expressions in Commands 154

Numeric Constants 155

Symbols 156

C Operators 159

8 Menu Bar Commands

Menu Bar Commands 162

File→Load Object... (ALT, F, L) 166
File→Command Log→Log File Name... (ALT, F, C, N) 168
File→Command Log→Logging ON (ALT, F, C, O) 169
File→Command Log→Logging OFF (ALT, F, C, F) 170
File→Run Cmd File... (ALT, F, R) 171
File→Load Debug... (ALT, F, D) 173
File→Save Debug... (ALT, F, S) 174
File→Load Emulator Config... (ALT, F, E) 175
File→Save Emulator Config... (ALT, F, V) 176
File→Copy Destination... (ALT, F, P) 177
File→Exit (ALT, F, X) 178
File→Exit HW Locked (ALT, F, H) 179
File Selection Dialog Boxes 180
Execution→Run (F5), (ALT, E, U) 181
Execution→Run to Cursor (ALT, E, C) 182
Execution→Run to Caller (ALT, E, T) 183
Execution→Run... (ALT, E, R) 184
Execution→Single Step (F2), (ALT, E, N) 186

Execution→Step Over (F3), (ALT, E, O) 187
Execution→Step... (ALT, E, S) 188
Execution→Break (F4), (ALT, E, B) 191
Execution→Reset (ALT, E, E) 192
Breakpoint→Set at Cursor (ALT, B, S) 193
Breakpoint→Delete at Cursor (ALT, B, D) 195
Breakpoint→Set Macro... (ALT, B, M) 196
Breakpoint→Delete Macro (ALT, B, L) 198
Breakpoint→Edit... (ALT, B, E) 199
Variable→Edit... (ALT, V, E) 203
Variable Modify Dialog Box 205
Trace→Function Caller... (ALT, T, C) 206
Trace→Function Statement... (ALT, T, S) 208
Trace→Variable Access... (ALT, T, V) 210
Trace→Edit... (ALT, T, E) 212
Trace→Trigger Store... (ALT, T, T) 213
Trace→Find Then Trigger... (ALT, T, D) 216
Trace→Sequence... (ALT, T, Q) 220
Trace→Until Halt (ALT, T, U) 224
Trace→Halt (ALT, T, H) 225
Trace→Again (F7), (ALT, T, A) 226
Condition Dialog Boxes 227
Trace Pattern Dialog Box 230
Trace Range Dialog Box 232
Sequence Number Dialog Box 234
RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) 235
RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) 236
RealTime→I/O Polling→ON (ALT, R, I, O) 237
RealTime→I/O Polling→OFF (ALT, R, I, F) 238
RealTime→Watchpoint Polling→ON (ALT, R, W, O) 239
RealTime→Watchpoint Polling→OFF (ALT, R, W, F) 240
RealTime→Memory Polling→ON (ALT, R, M, O) 241
RealTime→Memory Polling→OFF (ALT, R, M, F) 242
Assemble... (ALT, A) 243
Settings→Emulator Config→Hardware... (ALT, S, E, H) 244
Settings→Emulator Config→Memory Map... (ALT, S, E, M) 247
Settings→Emulator Config→Monitor... (ALT, S, E, O) 251
Settings→Emulator Config→Address Translations... (ALT, S, E, A) 254
Settings→Communication... (ALT, S, C) 258

Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O) 261
Settings→BNC→Input to Analyzer Arm (ALT, S, B, I) 262
Settings→Font... (ALT, S, F) 263
Settings→Tabstops... (ALT, S, T) 264
Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O) 265
Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F) 265
Settings→Extended Settings→Trace Cycles→User (ALT, S, X, T, U) 266
Settings→Extended Settings→Trace Cycles→Monitor (ALT, S, X, T, M) 266
Settings→Extended Settings→Trace Cycles→Both (ALT, S, X, T, B) 267
Settings→Extended Settings→Load Error Abort→ON (ALT, S, X, L, O) 268
Settings→Extended Settings→Load Error Abort→OFF (ALT, S, X, L, F) 268
Settings→Extended Settings→Source Path Query→ON (ALT, S, X, S, O) 269
Settings→Extended Settings→Source Path Query→OFF (ALT, S, X, S, F) 269
Window→Cascade (ALT, W, C) 270
Window→Tile (ALT, W, T) 270
Window→Arrange Icons (ALT, W, A) 270
Window→1-9 (ALT, W, 1-9) 271
Window→More Windows... (ALT, W, M) 272
Help→About Debugger/Emulator... (ALT, H, D) 273
Source Directory Dialog Box 274

9 Window Control Menu Commands

Window Control Menu Commands 276

Common Control Menu Commands 277

Copy→Window (ALT, -, P, W) 277
Copy→Destination... (ALT, -, P, D) 278

Button Window Commands 279

Edit... (ALT, -, E) 279

Expression Window Commands 282

Clear (ALT, -, R) 282
Evaluate... (ALT, -, E) 283

I/O Window Commands 284

Define... (ALT, -, D) 284

Memory Window Commands 286

Display→Linear (ALT, -, D, L) 286
Display→Block (ALT, -, D, B) 287
Display→Byte (ALT, -, D, Y) 287
Display→16 Bit (ALT, -, D, 1) 287
Display→32 Bit (ALT, -, D, 3) 287
Search... (ALT, -, R) 288
Utilities→Copy... (ALT, -, U, C) 290
Utilities→Fill... (ALT, -, U, F) 291
Utilities→Image... (ALT, -, U, I) 292
Utilities→Load... (ALT, -, U, L) 294
Utilities→Store... (ALT, -, U, S) 295

GDT/LDT/IDT Window Commands 297

Search→Entry... (ALT, -, R, E) 297
Search→Selector... (ALT, -, R, S) 298

Register Windows' Commands 300

Continuous Update (ALT, -, U) 300
Copy→Registers (ALT, -, P, R) 300

Register Bit Fields Dialog Box 301

Source Window Commands 303

Display→Mixed Mode (ALT, -, D, M) 303
Display→Source Only (ALT, -, D, S) 304
Display→Select Source... (ALT, -, D, L) 305
Search→String... (ALT, -, R, S) 306
Search→Function... (ALT, -, R, F) 307
Search→Address... (ALT, -, R, A) 309
Search Directories Dialog Box 310

Symbol Window Commands 311

Display→Modules (ALT, -, D, M) 311
Display→Functions (ALT, -, D, F) 312
Display→Externals (ALT, -, D, E) 312
Display→Locals... (ALT, -, D, L) 313
Display→Asm Globals (ALT, -, D, G) 314
Display→Asm Locals... (ALT, -, D, A) 314
Display→User defined (ALT, -, D, U) 316
Copy→Window (ALT, -, P, W) 316
Copy→All (ALT, -, P, A) 317
FindString→String... (ALT, -, F, S) 317
User defined→Add... (ALT, -, U, A) 318
User defined→Delete (ALT, -, U, D) 320
User defined→Delete All (ALT, -, U, L) 320

Trace Window Commands 321

Display→Bus Cycle ON (ALT, -, D, B) 322
Display→Source Only (ALT, -, D, S) 322
Display→Count→Absolute (ALT, -, D, C, A) 323
Display→Count→Relative (ALT, -, D, C, R) 323
Trace Display→From State... (ALT -, D, F) 324
Copy→Window (ALT, -, P, W) 325
Copy→All (ALT, -, P, A) 326
Search→Trigger (ALT, -, R, T) 326
Search→State... (ALT, -, R, S) 327
Trace Spec Copy→Specification (ALT, -, T, S) 328
Trace Spec Copy→Destination... (ALT, -, T, D) 328

WatchPoint Window Commands 329
Edit... (ALT, -, E) 329

10 Window Pop-up Menu Commands

Window Pop-up Menu Commands 334
BackTrace Window Pop-up Commands 335
Source at Stack Level 335

Source Window Pop-up Commands 336
Set Breakpoint 336
Clear Breakpoint 336
Evaluate It 337
Add to Watch 337
Run to Cursor 337

11 Other Command File and Macro Commands

Other Command File and Macro Commands 340
BEEP 341
EXIT 342
FILE CHAINCMD 343
FILE RERUN 344
NOP 345
TERMCOM 346
WAIT 347

12 Error Messages

Part 4 Concept Guide

13 Concepts

Concepts 362

Debugger Windows 363

The BackTrace Window 364
The Button Window 365
The Expression Window 366
The I/O Window 367
The Memory Window 369
The GDT Window 371
The LDT Window 373
The IDT Window 374
The Register Windows 375
The Source Window 376
The Status Window 379
The Symbol Window 382
The Trace Window 383
The WatchPoint Window 385

Monitor Program Options 386

Background monitor 387
Foreground monitor 387
Foreground monitor advantages and disadvantages 388

Trace Signals and Predefined Status Values 389

Understanding 80386 Analysis 391

Understanding Address, Data, and Status 395

Entering Addresses as Constants 398

Overview of 80386 address types 399
Explanation: why different syntax for different address types 400
Constant-address syntax 401

Unexpected Stepping Behavior 402

Faults 402
INT instructions 403
Task gates 403
To step into a task or a fault handler 403

Part 5 Installation Guide

14 Installing the Debugger

Installing the Debugger 408

Requirements 409

Before Installing the Debugger 410

Step 1. Connect the HP 64700 to the PC 411

To connect via RS-232 411
To connect via LAN 414
To connect via RS-422 418
If you cannot verify RS-232 communication 419
If you cannot verify LAN communication 420

Step 2. Install the debugger software 421

Step 3. Start the debugger 424

If you have RS-232 connection problems 424
If you have LAN connection problems 426
If you have LAN DLL errors 427
If you have RS-422 connection problems 428

Step 4. Check the HP 64700 system firmware version 429

Optimizing PC Performance for the Debugger 430

15 Installing/Updating HP 64700 Firmware

Installing/Updating HP 64700 Firmware 432

Step 1. Connect the HP 64700 to the PC 433
Step 2. Install the firmware update utility 435
Step 3. Run PROGFLASH to update HP 64700 firmware 437

Glossary

Index

Part 1

Quick Start Guide

A few task instructions to help you get comfortable.

1

Getting Started

Getting Started

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

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

Demo Programs

Demo programs are included with the Real-Time C Debugger in the C:\HP\RTC\I386\DEMO directory (if C:\HP\RTC\I386 was the installation path chosen when installing the debugger software).

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

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

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

Step 1. Start the debugger

1 Cycle power on the HP 64700-Series Card Cage to ensure that the emulator will be in its default state when you begin this tutorial. Wait a minute to allow time for the boot-up routine to complete.

2 Open the HP Real-Time C Debugger group box and double-click the 80386 debugger icon.

Or:

3 Choose the File→Run (ALT, F, R) command in the Windows Program Manager.

4 Enter the debugger startup command, C:\HP\RTC\I386\B3637.EXE (if C:\HP\RTC\I386 was the installation path chosen when installing the debugger software).

5 Choose the OK button.

Step 2. Adjust the fonts and window size

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

1. Choose the Settings→Font... (ALT, S, F) command.
2. Choose the Font, Font Style, and Size desired in the Font dialog box.
3. Choose the OK button to apply your selections, and close the Font dialog box.

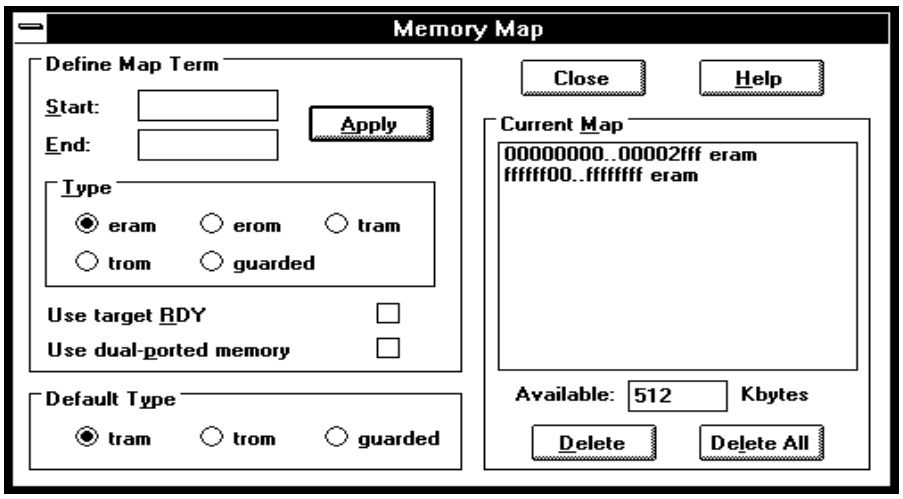
The sizes of the RTC window, as well as the sizes of the windows within RTC, and the fonts used will be saved in the BXXXX.INI file and reused when you enter RTC the next time.

Step 3. Map memory for the demo program

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

The demo sample program occupies address ranges 0h-2fffh and 0ffffff00h-0ffffffffh. Map these address ranges in emulation RAM memory.

1. Choose the Settings→Emulator Config→Memory Map... (ALT, S, E, M) command.
2. Enter "0" in the Start text box.
3. Tab the cursor to the End text box and enter "2fff".
4. Select "eram" in the Type option box.
5. Unselect Use target RDY and leave Use dual-ported memory unselected.
6. Choose the Apply button.
7. Enter "0ffffff00" in the Start text box, enter "0ffffffff" in the End text box. Select "eram" in the Type option box for this range also, and choose the Apply button.
8. Choose the Close button.



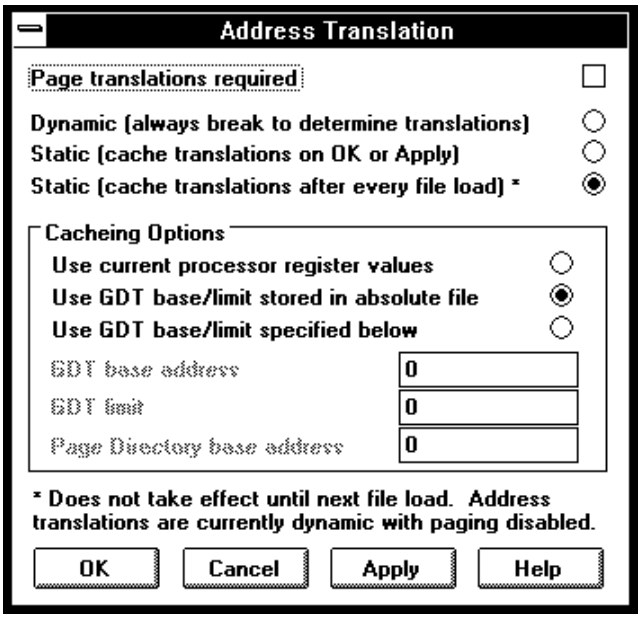
Step 4. Set address translations for the demo program

1 Choose the Settings→Emulator Config→Address Translation... (ALT, S, E, A) command.

2 Set up the Address Translation dialog box as shown in the illustration.

3 Choose the OK button.

This is the default setup for the Address Translation dialog box. It ensures that the emulator can refer to protected-mode addresses (for setting breakpoints) before running the demo program.



Step 5. Load the demo program

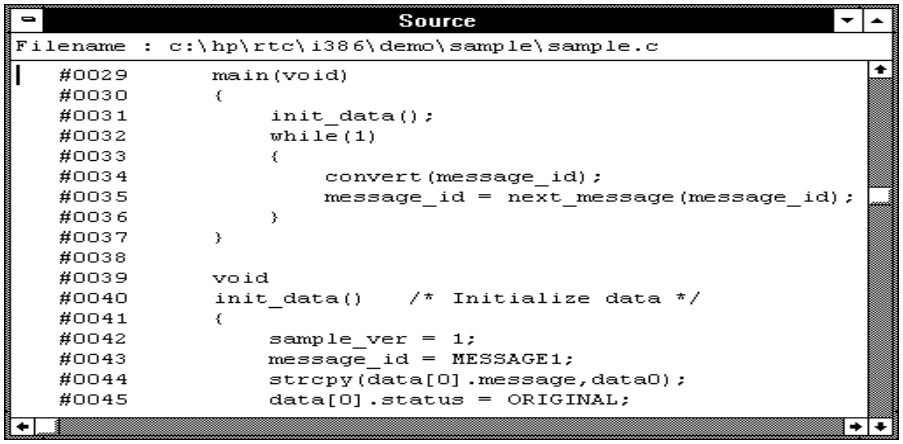
1. Choose the Execution→Break (ALT, E, B) command.
2. Choose the File→Load Object... (ALT, F, L) command.
3. Choose the Browse button and select the sample program object file, C:\HP\RTC\I386\DEMO\SAMPLE\SAMPLE (if C:\HP\RTC\I386 was the installation path chosen when installing the debugger software).
4. Choose the OK button in the Object File Name dialog box.
5. Choose the Load button.

Step 6. Display the source file

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

1. If the Source window is not open, double-click on the Source window icon to open the window. Or, choose the Window→Source command.
2. From the Source window's *control menu*, choose Search→Function... (ALT, -, R, F) command.
3. Select "main".
4. Choose the Find button.
5. Choose the Close button.
6. From the Source window's *control menu*, choose Display→Source Only (ALT, -, D, S) command.

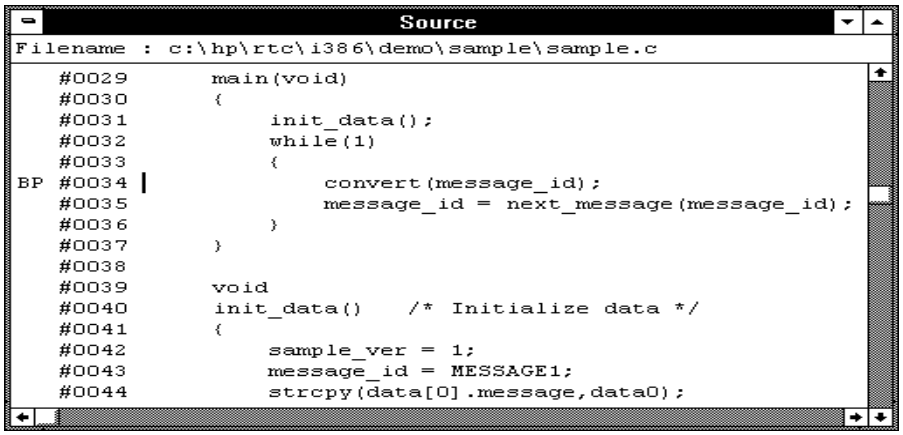
   The window displays sample.c source file, starting from main function.



Step 7. Set a breakpoint

To set a breakpoint on line 34 in sample.c:

1. Cursor-select line 34.
2. Choose the Breakpoint→Set at Cursor (ALT, B, S) command.



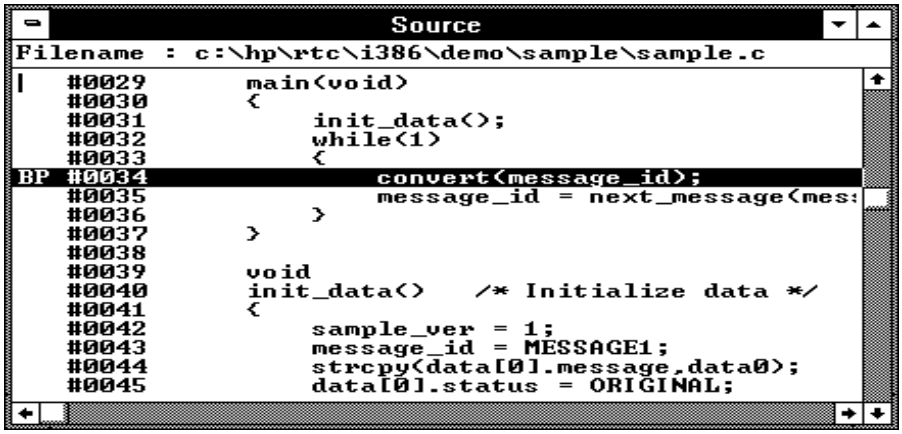
Notice that line 34 is marked with "BP" which indicates a breakpoint has been set on the line.

Step 8. Run the demo program

To run the demo program from the reset address:

1. Choose the Execution→Run... (ALT, E, R) command.
2. Select the User Reset option.
3. Choose the Run button.

Notice the demo program runs until line 34. The highlighted line indicates the current program counter.



Step 9. Delete the breakpoint

To delete the breakpoint set on line 34:

1. Cursor-select line 34.
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.

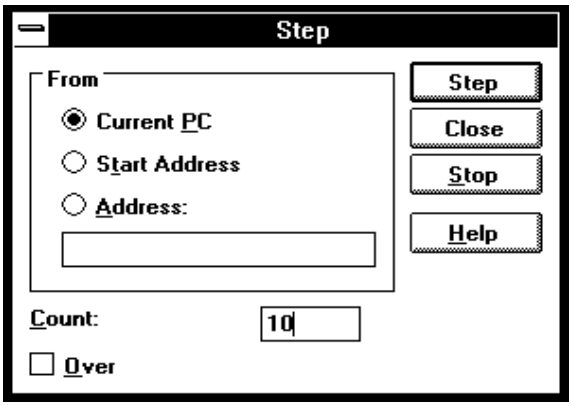
Step 11. Single-step 10 lines

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.

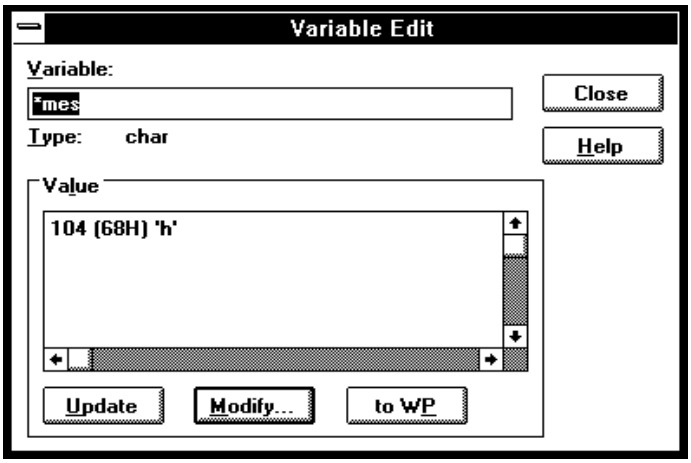
5 Choose the Close button.

Step 12. Display a variable

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

1 Drag "*mes" on line 60 in the Source window until it is highlighted.

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



The Variable text box displays "*mes".

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

Note

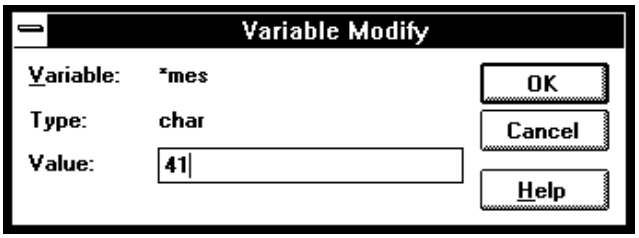
You can only register or display an auto variable as a watchpoint while the program counter is within the function in which the variable name is declared.

Step 13. Edit a variable

To edit the contents of variable "*mes":

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

2 Enter "41" in the Value text box.



3 Choose the OK button.

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

Step 14. Monitor a variable in the WatchPoint window

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

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

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

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

Step 15. Run until return from current function

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

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

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

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.

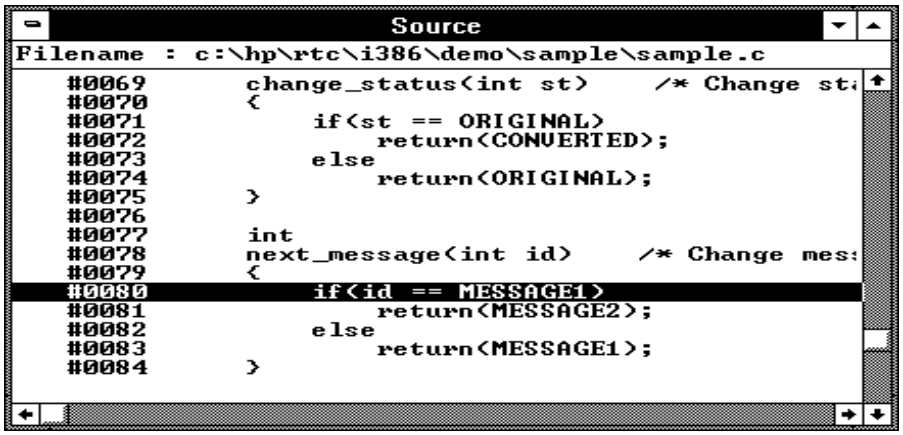
  The "change_status" function executes, and the program counter indicates line 55.

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 80.

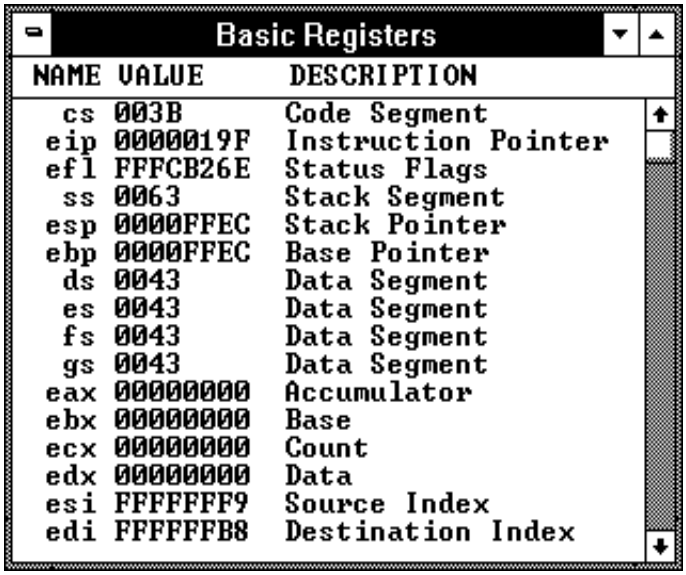
2 Choose the Execution→Run to Cursor (ALT, E, C) command.



The program executes and stops immediately before line 80.

Step 18. Display register contents

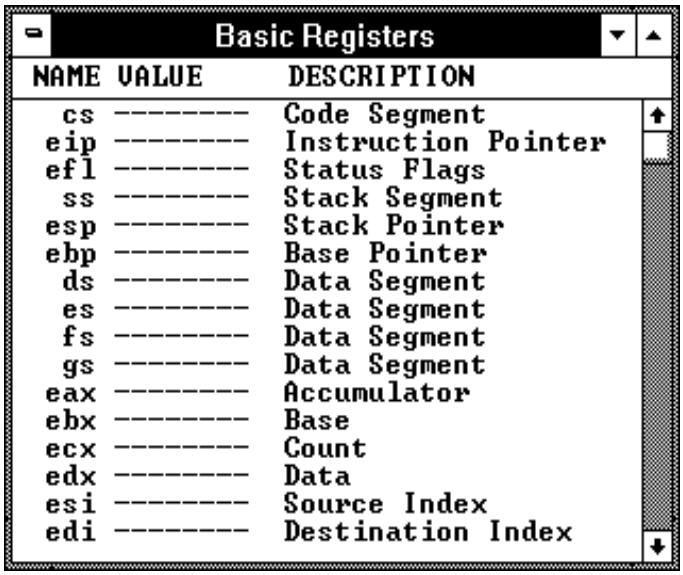
1 Choose the Window→Basic Registers command.



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

2 To prevent the register display from being updated, choose the RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command.

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



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

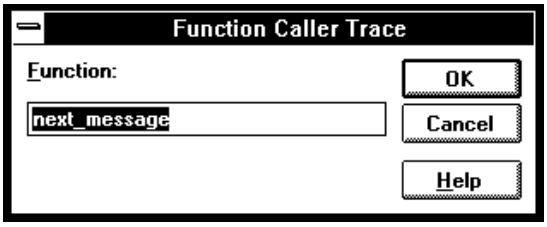
4 Choose the RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command to deselect the real-time mode. Notice that the contents of the registers are updated periodically.

Step 19. Trace a function's callers

To trace the caller of "next_message":

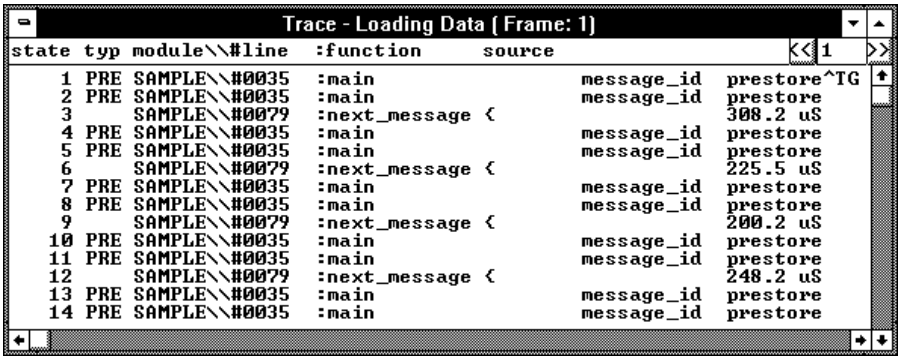
1 Double-click "next_message" on line 78 in the Source window.

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



3 Choose the OK button.

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



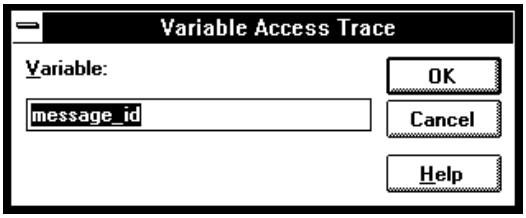
This command stores the first statement of a function and prestores statements that occur before the first statement (notice the state type PRE). The prestored statements show the caller of the function. In the above example, "next_message" is called by line 35 of "main".

Step 20. Trace access to a variable

To trace access to variable "message_id":

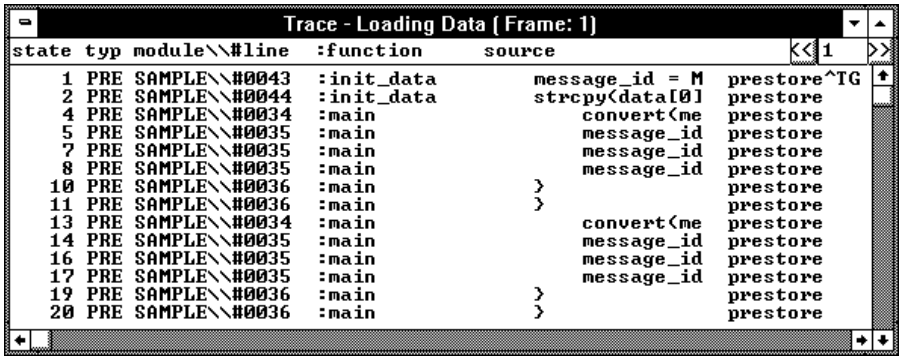
1 Double-click "message_id" in the Trace window or on line 35 in the Source window.

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



3 Choose the OK button.

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



Line 35 displays three times because it accessed "message_id" twice for reads and once for a write.

Step 21. Exit the debugger

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

   This will end your Real-Time C Debugger session.

Part 2

User's Guide

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

2

Using the Debugger Interface

Using the Debugger Interface

This chapter contains general information about using the debugger interface.

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

How the Debugger Uses the Clipboard

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

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

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

Debugger Function Key Definitions

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

Starting and Exiting the Debugger

This section shows you how:

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

To start the debugger

- Double-click the debugger icon.

  Or:

1. Choose the File→Run (ALT, F, R) command in the Windows Program Manager.
2. Enter the debugger filename, C:\HP\RTC\I386\B3637.EXE (if C:\HP\RTC\I386 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.

   In the Command Line text box, enter the "C:\HP\RTC\I386\B3637.EXE -T<transport> -E<connectname>" command (if C:\HP\RTC\I386 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 B3637.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.
5. In the Working Directory text box, enter the directory that contains the debugger program (for example, C:\HP\RTC\I386).
6. 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 tab stops in the Source window

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

2 Enter the tab width. This width is also used for source lines in the trace window.

3 Choose the OK button.

Using Command Files

This section shows you how:

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

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

To create a command file

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

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

To execute a command file

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

2 Select the command file to be executed.

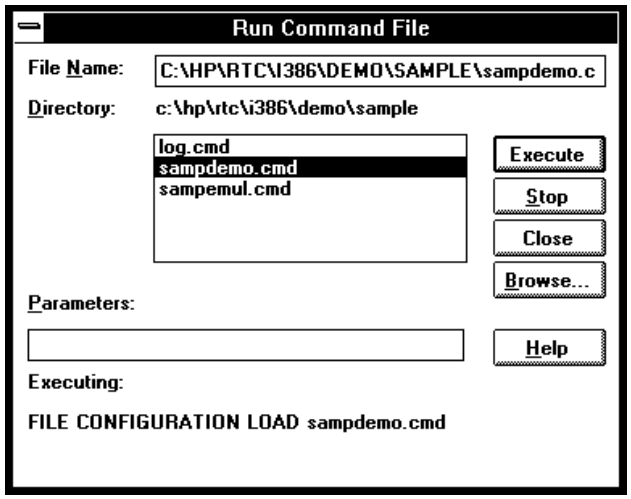
3 Choose the Execute button.

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

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

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

Example Command File Being Executed



To create buttons that execute command files

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

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

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

3

Plugging the Emulator into Target Systems

Plugging the Emulator into Target Systems

This chapter shows you how:

- Step 1. Turn OFF power
- Step 2. Unplug probe from demo target system
- Step 3. Plug the probe into the target system
- Step 4. Connect the reset flying lead to the target system
- Step 5. Turn ON power

CAUTION

Possible Damage to the Emulator Probe. The emulation probe contains devices that are susceptible to damage by static discharge. Take precautionary measures before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe by static electricity.

HP STRONGLY recommends you use a ground strap when handling the emulator probe. A ground strap is provided with the emulator.

There is a red LED on the probe board. If the LED is on, immediately turn off your target system! The LED turns on when your target system has power but the emulator does not. Permanent damage will occur if target system power is turned on when the emulator is turned off, especially if this condition lasts for more than one minute.

Step 1. Turn OFF power

CAUTION

Possible Damage to the Emulator. Make sure target system power is OFF and make sure HP 64700 power is OFF before removing or installing the emulator probe into the target system.

Do not turn HP 64700 power OFF while the emulator is plugged into a target system whose power is ON.

1 If the emulator is currently plugged into a different target system, turn that target system's power OFF.

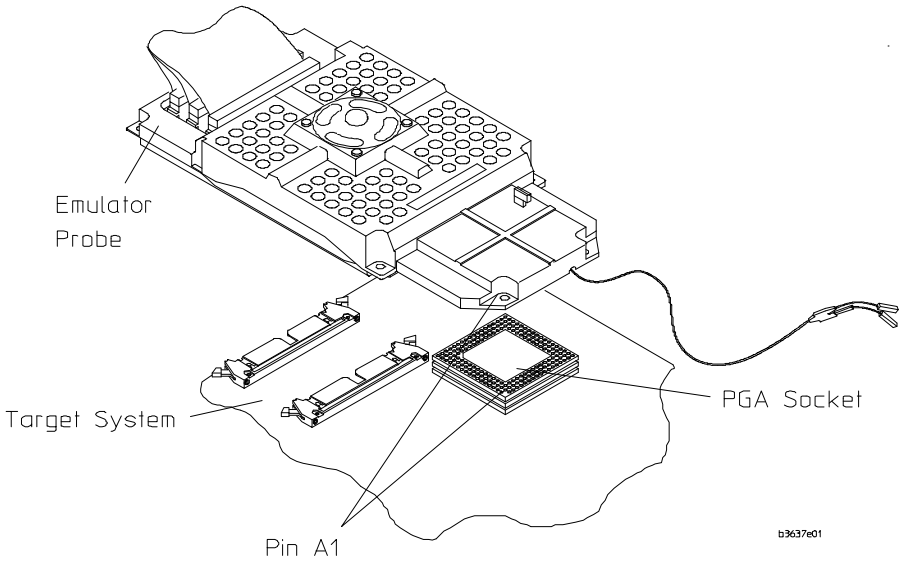
2 Turn emulator power OFF.

Step 2. Unplug the probe from the demo target system

- If the emulator is currently connected to a different target system, unplug the emulator probe; otherwise, disconnect the emulator probe from the demo target system.

Step 3. Plug the probe into the target system

- Install the emulator probe into the target system socket. Make sure that pin A1 of the connector aligns with pin A1 of the socket. **Damage to the emulator will result if the probe is incorrectly installed.**



You can also use the supplied PGA to PGA adapter or other PQFP adapters. Always make sure that pin 1 and other pins of the adapters and connectors are properly aligned; otherwise, damage to the emulator will result.

Step 4. Connect the reset flying lead to the target system

- The details of how to connect the reset flying lead are shown in the HP 64789A i80386 Emulator Installation/Service/Terminal Interface User's Guide.

The reset flying lead on the 80386 emulator can be used to reset your target system when the emulator applies reset to the processor. This is useful if you have any hardware in your target system that needs to know when the processor is reset (such as a circuit to generate the self-test request to the processor).

The reset flying lead is an open-collector circuit that will go low when the emulator applies reset (that is, you have used the "reset" command, have reconfigured the emulator, or have given any other command that results in the processor being reset.) It will not go low when your target system applies reset unless the emulator is also applying reset.

You do not need to use this if the only signal your target system derives from RESET is the "CLK" signal; the emulator will preserve the phase of CLK between emulation-only resets.

Step 5. Turn ON power

1 Turn emulator power ON.

2 Turn target system power ON.

4

Configuring the Emulator

Configuring the Emulator

This chapter contains information about configuring the emulator.

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

Setting the Hardware Options

This section shows you how:

- To specify a CLK2 speed faster than 60 MHz
- To enable or disable target interrupts
- To enable or disable software breakpoints
- To enable or disable break on writes to ROM
- To enable or disable execution trace messages
- To enable or disable foreground monitor traced as user

To specify a CLK2 speed faster than 60 MHz

1. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.
2. Select or deselect the Processor Clock is Faster Than 60 MHz check box.
3. Choose the OK button to exit the Emulator Configuration dialog box.

If the 4-Mbyte SIMMs are installed, and the CLK2 speed is greater than 60 MHz, the emulator has to force at least one wait state because the 4-MByte SIMMs are slower than the 256-KByte and 1-Mbyte SIMMs.

CLK2 is the clock input to the 80386; it is twice the speed of the usually-quoted speed (that is, a "25 MHz 80386" has a CLK2 speed of 50 MHz).

Note that if you lock emulation memory cycles with target cycles, the target hardware must continue to assert the READY# line until the second wait state.

To enable or disable target interrupts

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

2 Select or deselect the Enable Target Interrupts check box.

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

If selected, the emulator responds to interrupts generated by the target system while running in the user program or foreground monitor. All interrupts (INT or NMI) are blocked when execution is within the background monitor.

If deselected, the emulator ignores all interrupts generated by the target system, INT and NMI.

To enable or disable software breakpoints

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

2 Select or deselect Enable Software Breakpoints check box.

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

If selected, the processor will take longer to leave the RESET state than when breakpoints are disabled.

The emulator uses the bond-out processor's software breakpoint capability. This requires a special bit to be set to enable recognition of the breakpoint instruction (which is a special opcode, different from the normal execution breakpoint opcode of 0CCH). When the processor is reset, this bit is cleared. To make use of breakpoints, the emulation monitor must set this bit every time the processor leaves the reset state.

To enable or disable break on writes to ROM

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

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

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

If selected, a running program breaks into the monitor when it writes to a location mapped as ROM.

If deselected, program writes to locations mapped as ROM do not cause breaks into the monitor.

To enable or disable execution trace messages

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

2 Select or deselect the Enable Execution Trace Messages check box.

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

If selected, branch trace messages and task switch messages are enabled. Every time the processor does a tranch, it will emit the target address of the branch. Each time a task switch occurs, the emulator will emit a task switch message identifying both the old task and the new task.

If deselected, no branch trace messages nor task switch messages will be emitted.

To enable or disable foreground monitor traced as user

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

2 Select or deselect the Enable Monitor Traced as User check box.

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

If selected when using a foreground monitor, all foreground monitor cycles will be captured in the trace memory by the emulation-bus analyzer. This is useful when you are having problems with an interrupt routine and you want to trace that routine even if it occurs during execution in the foreground monitor.

If deselected and you have chosen Settings→Extended→Trace Cycles→User, the analyzer will capture nothing between the time the foreground monitor is entered and the time you begin a run of your user program again. This prevents capture of interrupt routines executed while in the foreground monitor. This is useful when you are trying to conserve trace memory space to capture user program execution.

When using the background monitor, this has no effect.

See "Tracing Program Execution" in the "Debugging Programs" chapter for useful combinations of the "Settings→Extended→Trace Cycles" command and the Enable Foreground Monitor Traced as User selection.

Selecting the Type of Monitor

This section shows you how:

- To select the background monitor
- To select the foreground monitor
- To use a custom 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.

To select the foreground monitor

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

2 Select the Foreground option.

3 Enter the base address of the foreground monitor in the Monitor Address text box. The address must reside on a 16 Kbyte boundary (in other words, the address must be a multiple of 4000H) and must be specified in hexadecimal.

4 Enter the GDT descriptor for the foreground monitor code segment. This reserves a GDT entry to define the code segment for the monitor when running in protected mode. The specified value must be a multiple of 8, greater than 0 and less than the limit defined in GDTR.

5 If you wish to synchronize monitor cycles to the target system (that is, interlock the emulation and target system READY# lines on accesses to the monitor memory block), select the Monitor Cycles Use Target RDY option; otherwise, deselect this option.

6 Leave the Load Custom Monitor box unselected. This tells the emulator to use the default foreground monitor present in the emulator firmware.

7 Choose the OK button.

8 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 you select the foreground monitor, the emulator automatically loads the default foreground monitor program, resident in emulator firmware, into emulation memory. The foreground monitor is reloaded every time the emulator breaks into the monitor state from the reset state.

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

To use a custom foreground monitor

1 Edit the foreground monitor program source.

2 Assemble and link the foreground monitor program.

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

4 Select the Foreground option.

5 Enter the base address of the foreground monitor in the Monitor Address text box. The address must reside on a 16-Kbyte boundary (an address ending in 4000H) and must be specified in hexadecimal.

6 If you wish to synchronize monitor cycles to the target system (that is, interlock the emulation and target system READY# lines on accesses to the monitor memory block), select the Monitor Cycles Use Target RDY option; otherwise, deselect this option.

7 Enter the name of the foreground monitor object file in the Monitor File 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 and adds a new map term for the monitor.

10 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 customizing the foreground monitor, you must maintain the basic communication protocol between the monitor program and the emulation system controller.

An example foreground monitor is provided with the debugger in the \HP\RTC\I386\MONITOR directory (if that is the directory where the software was installed). The file is named I386DX.ASM.

The custom foreground monitor is saved in the emulator (until the monitor type is changed) and reloaded every time the emulator breaks into the monitor state from the reset state.

Mapping Memory

This section shows you how:

- To map memory

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

There are two types of emulation memory: SIMMs, and dual-port memory. 256-Kbyte, 1-Mbyte, and 4-Mbyte SIMMs are supported, although the 4-Mbyte SIMMs require an additional wait state if the CLK2 speed in your target system is greater than 60 MHz.

The dual-port memory is 8 Kbytes and is always available (even when using a foreground monitor). The differences between dual-port memory and SIMM memory are:

- Dual port memory is always available, even when no SIMMs are loaded
- Only one map term (address range) can be used with the dual-port attribute
- The user interface can access data stored in dual-port RAM without interrupting any programs running on the 80386. If the processor is executing instructions, the memory is access transparently by interleaving accesses from the 80386 with accesses from the emulator. If the processor is RESET, or there is no power to the target system, the dual-port memory can be accessed normally (transparently). If the processor is in the HALT or SHUTDOWN state, however, dual-port memory cannot be accessed transparently. In that case, the monitor will be used. To prevent the monitor from being used, choose Realtime→Monitor Intrusion→Disallowed.

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

Note that the if you have a 1-Mbyte SIMM, but you map all eight terms to 256-byte segments (for a total of 2 Kbytes), the remaining 1022 Kbytes cannot be used.

External direct memory access (DMA) to emulation memory is not permitted.

You should map all memory ranges used by your programs 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 or deselect the Use Target RDY option.
5. Select or deselect the Use dual-ported memory option.
6. Select the memory type in the Type option box.
7. Choose the Apply button.
8. Repeat steps 2 through 7 for each range to be mapped.
9. Choose the Close button to exit the Memory Map dialog box.

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".

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

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.

Writes to emulation ROM will modify memory. Writes by user code to target system memory locations that are mapped as ROM or guarded memory may result in a break to the monitor but they are not inhibited (that is, the write still occurs).

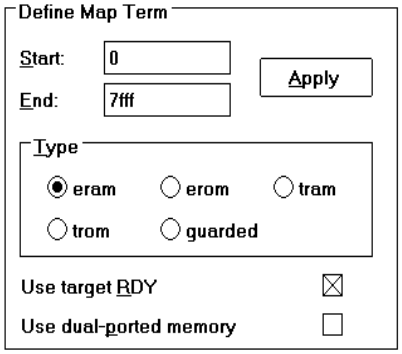
The Use Target RDY option specifies that emulation memory accesses in the range be synchronized to the target system RDY signal.

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.

Example

To map addresses 0 through 7fffh 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.

Selecting Address Translations

1. Choose the Settings→Emulator Config→Address Translations... (ALT, S, E, A) command.
2. Leave Page translations required unselected unless your target system uses paging.
3. Select the method of determining translations.
4. If you selected one of the static methods of determining translations, select the desired Cacheing Option.
5. Choose the OK button to apply your selections and close the Address Translation dialog box, or choose Apply to apply your selections and leave the dialog box open on screen.

When address translations occur

Translations are necessary whenever a request is made to access target or emulation memory (such as displaying memory or modifying memory), or whenever a trace is set up.

If paging is not being used, it is not necessary to break processor execution in order to translate a real-mode address. If paging is being used, processor execution must be broken (because the real-mode address may be a virtual-8086 address).

Implications of address translation options

The method used to translate addresses determines the accuracy and intrusiveness of the emulator:

Dynamic translations cause a temporary break (from your program into the monitor) to do a translation. This means that the translation is always accurate for the current state of the processor and for the current GDT (if in protected mode).

If your GDT tables change frequently, dynamic translation may be the best option for you; however, you cannot set up the analyzer or modify and display memory using protected-mode addresses when the processor is RESET. You must use physical addresses in that case.

Static translations cache the GDT and LDT tables (either from a program or from the current tables in the processor), and use the cached values of the tables to translate all virtual addresses. Static translations are only accurate if the current GDT matches the cached GDT values. When using Static translations, your program is never interrupted in order to perform a translation. You can use protected-mode addresses while the processor is RESET (or in real mode) to modify and display memory or set up a trace.

Page translations can also be dynamic or cached. Note that if you do not use paging, you will obtain better performance by turning off the check box next to "Page translations required".

Setting Up the BNC Port

This section shows you how:

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

To output the trigger signal on the BNC port

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

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

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

To receive an arm condition input on the BNC port

- **Choose the Settings→BNC→Input to Analyzer Arm (ALT, S, B, 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 (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, I/O, and microprocessor registers are displayed or edited. In addition, any address translations will cause the monitor program to execute unless they are configured to be static (see Selecting how Address Translations work). Also, periodic polling to update the Memory, I/O, WatchPoint, and Register windows can cause monitor program execution.

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 is 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.

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

When monitor intrusion is allowed, you can turn OFF polling for particular windows 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.

5

Debugging Programs

Debugging Programs

This chapter contains information on loading and debugging programs.

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

Loading and Displaying Programs

This section shows you how:

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

To load user programs

1 Choose the File→Load Object... (ALT, F, L) command.

2 Select the file to be loaded.

3 Choose the Load button to load the program.

With this command, you can load any Intel OMF object file created with any of the Microtec or HP programming tools for 80386.

To display source code only

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

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

   The display starts from the line containing the cursor.

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

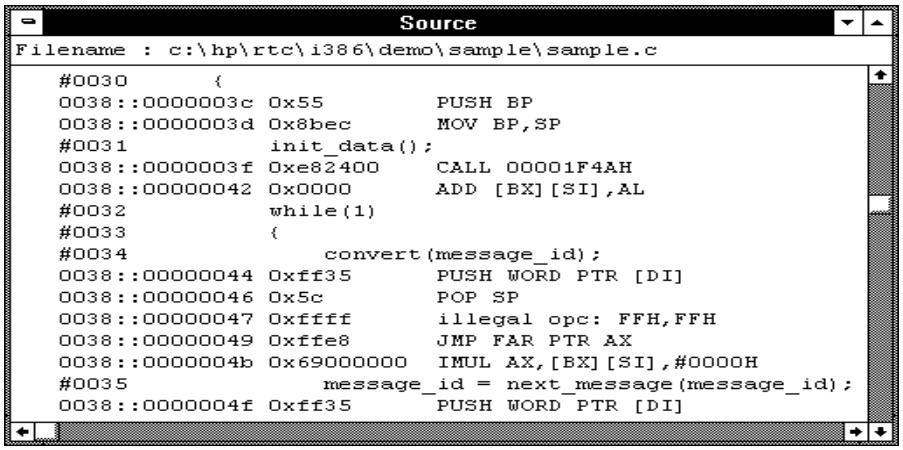
To display source code mixed with assembly instructions

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

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

Example

C Source/Mnemonic Mode Display



To display source files by their names

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

Note

The contents of assembly language source files cannot be displayed.

To specify source file directories

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

If the source files associated with the loaded object file are in different directories 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

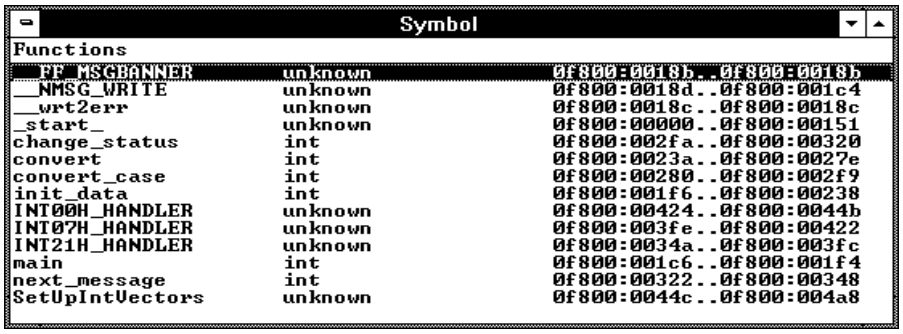
- From the Symbol window's control menu, choose the Display→Modules (ALT, -, D, M) command.

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



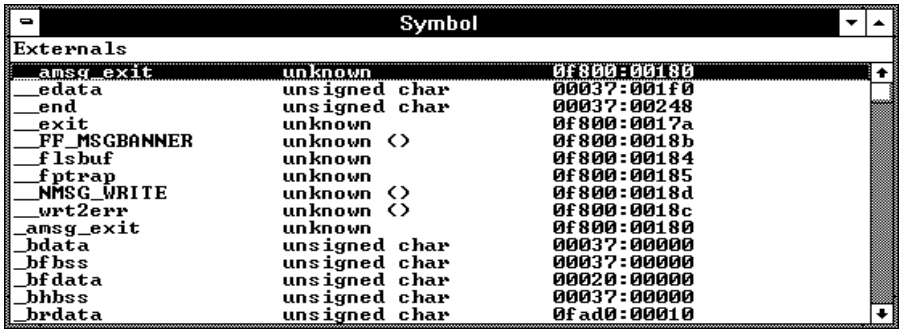
To display external symbol information

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

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

Example

External Symbol Information Display



To display local symbol information

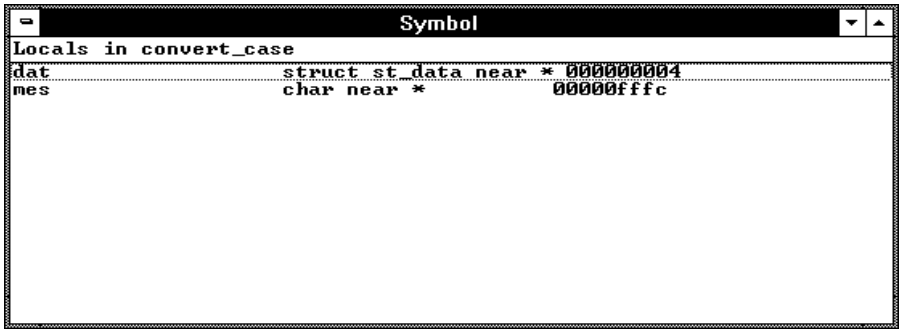
1 From the Symbol window's control menu, choose the Display→Locals... (ALT, -, D, L) command.

2 Type or paste the function for which the local variable information is to displayed.

3 Choose the OK button.

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

Example Local Symbol Information Display



To display global assembler symbol information

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

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

To display local assembler symbol information

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

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

To create a user-defined symbol

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

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

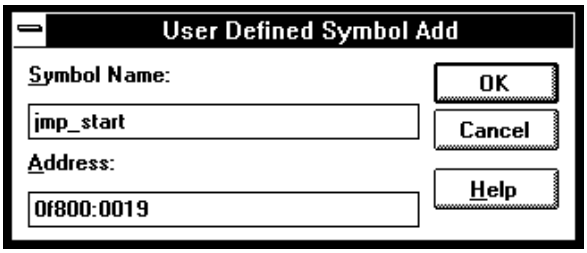
3 Type the address in the Address text box.

4 Choose the OK button.

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

Example

To add the user-defined symbol "jmp_start":

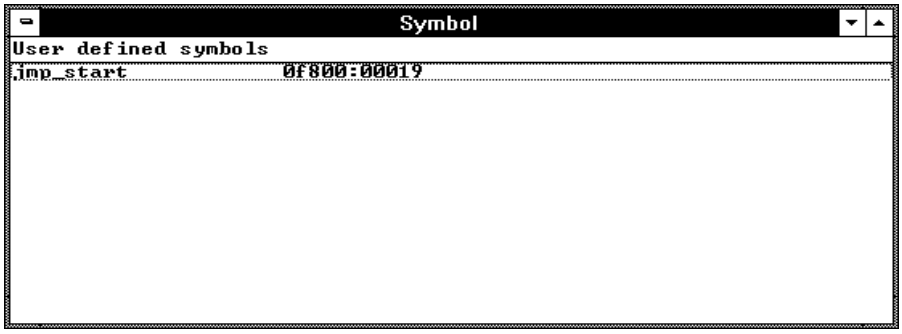


To display user-defined symbol information

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

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

Example User-Defined Symbol Information Display



To delete a user-defined symbol

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

To display the symbols containing the specified string

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

To restore the original nonselective display, redisplay the symbolic information.

Stepping, Running, and Stopping the Program

This section shows you how:

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

To step a single line or instruction

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

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

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

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

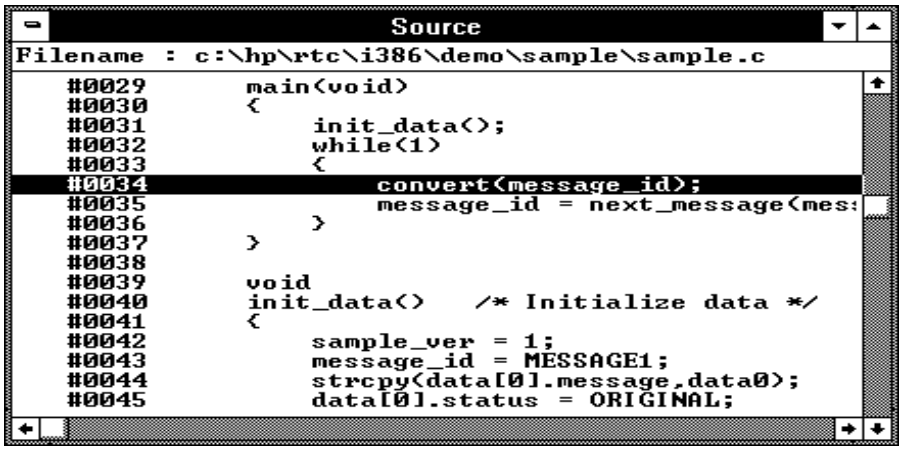
During a single-step command, multiple instructions can be executed if the instruction being stepped causes an instruction fault or task switch. See "Unexpected Stepping Behavior" in the "Concepts" chapter.

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 34, 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 35.

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 if you are already using the four hardware breakpoints on the 80386 and the address you are stepping is in target ROM.

If the specified address is not reached within the number of milliseconds specified by StepTimerLen in the B3637.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 from the reset vector. Note that this will cause your target board to reset only if you have attached the "reset flying lead" to the appropriate spot in your target system.

The Address option executes the program from the address specified.

See Also

"Step 4. Connect the reset flying lead to the target system" in the "Plugging the Emulator into Target Systems" chapter.

To stop program execution

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

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

To reset the processor

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

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

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

Using Breakpoints and Break Macros

This section shows you how:

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

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).

You may have any number of "software breakpoints", which are set by replacing opcodes in the program.

You may have up to four "hardware breakpoints", which are breakpoints for code that is in target system ROM.

To set a breakpoint

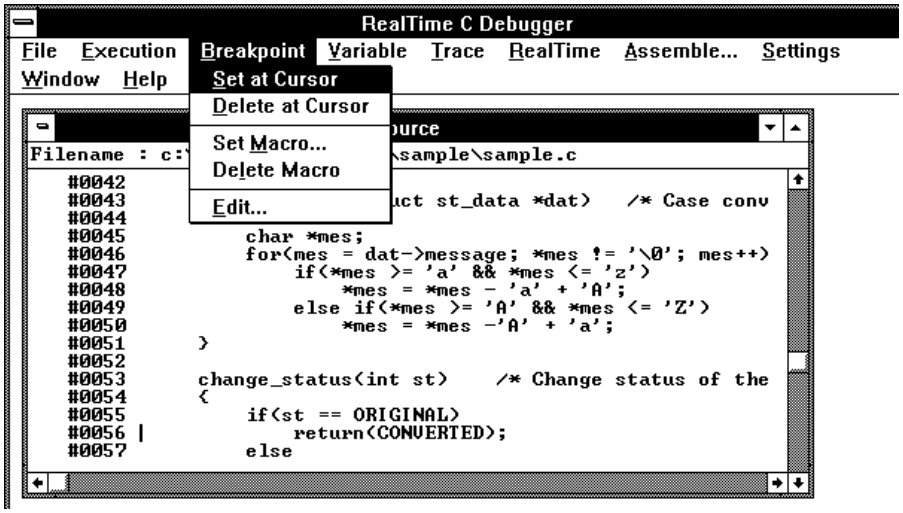
1 Position the cursor on the line where you wish to set a breakpoint.

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 56:



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. Notice that "DI" appears next to the breakpoint in the list.
4. To close the dialog box, choose the Close button.

You can reenable a breakpoint in the same manner by choosing the Breakpoint→Edit... (ALT, B, E) command, selecting a disabled breakpoint from the list, and deselecting the 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. Specify the macro command in the Macro Command text box.
4. Choose the Insert button.
5. To add another macro command, repeat steps 3 and 4.
6. 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).

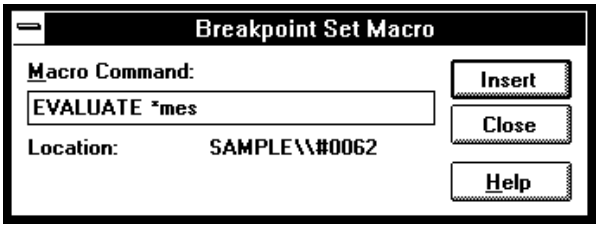
To add macro commands to the top of an existing break macro, position the cursor on the line that contains the BP marker before choosing Breakpoint→Set Macro... (ALT, B, M).

Example

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

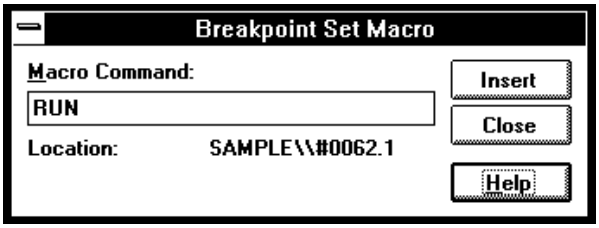
Position the cursor on line 62; then, choose the Breakpoint→Set Macro... (ALT, B, M) command.

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



Choose the Insert button.

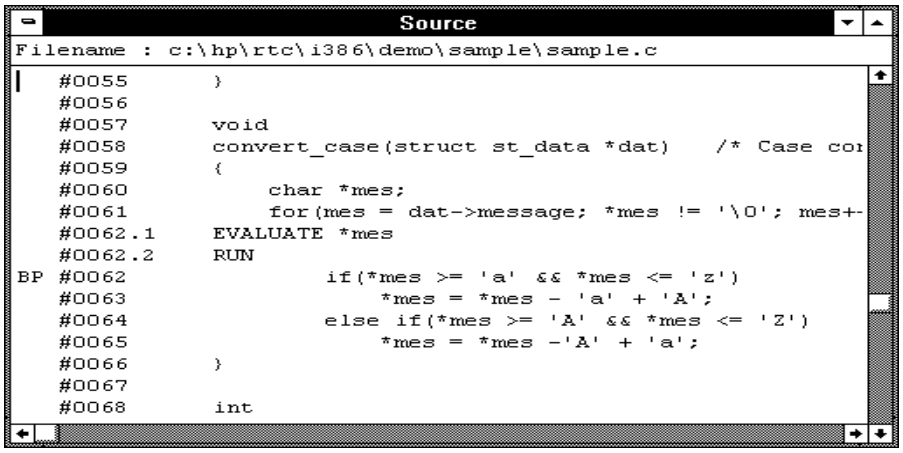
Enter "RUN" in the Macro Command text box.



Choose the Insert 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.

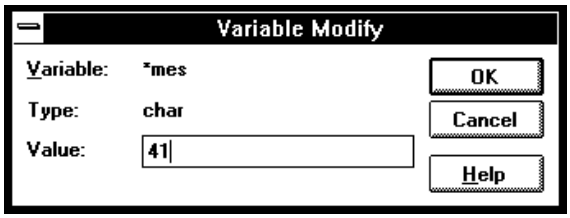
To edit a variable

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

To display memory

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

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

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

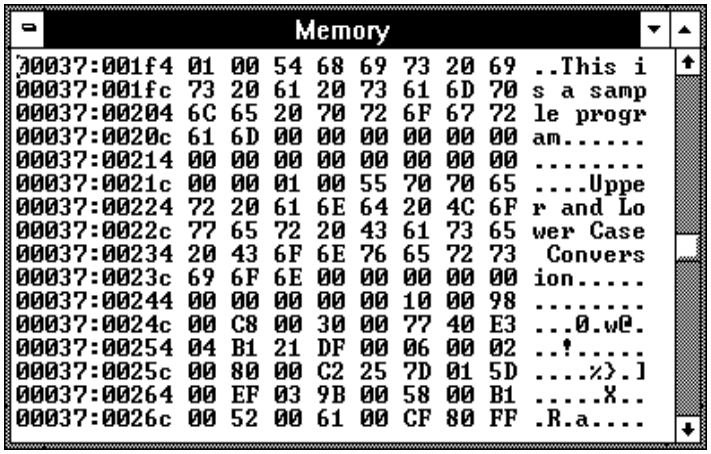
Display→Byte (ALT, -, D, Y) command is chosen, ASCII values are also displayed.

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

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

Example

Memory Displayed in Byte Format



To edit memory

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

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

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

To copy memory to a different location

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

To copy target system memory into emulation memory

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

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

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

4 Enter the starting address in the Start text box.

5 Enter the end address in the End text box.

6 Choose the Execute button.

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

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

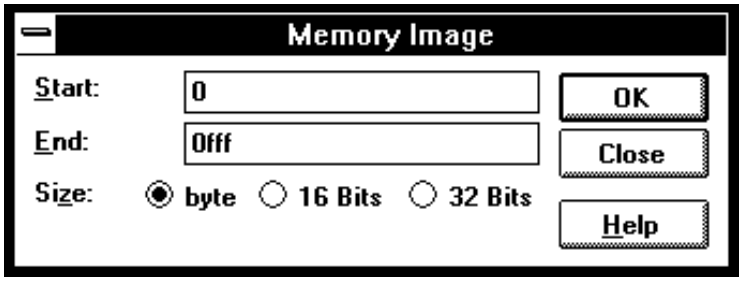
If you want to have more than four breakpoints in target system ROM, you may use the Utilities→Image... command to copy the memory and use software breakpoints instead of the four hardware breakpoints.

Note that the following commands use breakpoints:

- 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)

Example

To copy the contents of addresses 0 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, 16 Bit, or 32 Bit size option specifies the size of the values that are used to fill memory.

To search memory for a value or string

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

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

3 Enter the starting address in the Start text box.

4 Enter the end address in the End text box.

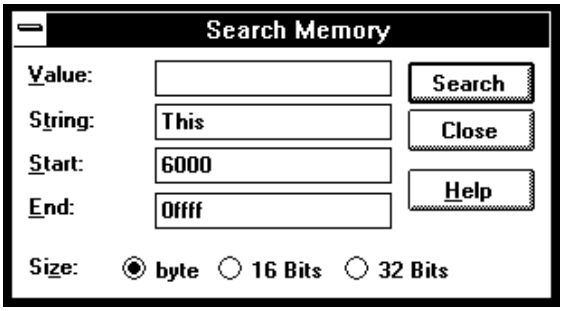
5 Choose the Execute button.

6 Choose the Close button.

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

Example

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



Displaying and Editing GDT, LDT, and IDT Windows

This section shows you how:

- To display the GDT, LDT, and IDT windows
- To edit the GDT, LDT, and IDT windows

To display the GDT, LDT, and IDT windows

- **Choose the Window→GDT, Window→LDT, or the Window→IDT, command.**

  The Window→GDT, Window→LDT, and Window→IDT commands display the contents of the specified window.

  The debugger periodically reads the GDT, LDT, and IDT locations and displays the latest content in the selected window.

To edit the GDT, LDT, and IDT windows

1. Choose the Window→GDT, Window→LDT, or Window→IDT command.
2. Find the physical address associated with the value to be changed.
3. Display the Memory window with the Window→Memory command.
4. Find the same physical address in the Memory window that you found in the GDT, LDT, or IDT window.
5. Use the keyboard to modify the content associated with the physical address, as desired.
6. Press the Enter key. Notice that the next location is highlighted.

   As long as the cursor remains in the Memory window, the GDT, LDT, or IDT window will not show your new value. Move the cursor out of the Memory window to see the GDT, LDT, or IDT window update to the new value.

Displaying and Editing I/O Locations

This section shows you how:

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

To display I/O locations

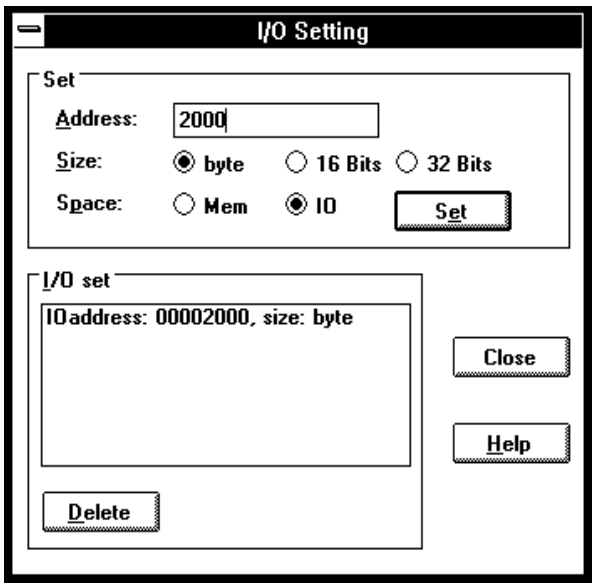
1. Choose the Window→I/O command.
2. From the I/O window's control menu, choose the Define... (ALT, -, D) command.
3. Enter the address in the Address text box.
4. Select whether the size of the I/O location is a Byte, 16 Bits, or 32 Bits.
5. Select whether the I/O location is in Memory or I/O space.
6. Choose the Set button.
7. Choose the Close button.

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

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

Example

To display the contents of address 2000:



To edit an I/O location

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

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

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

Displaying and Editing Registers

This section shows you how:

- To display registers
- To edit registers

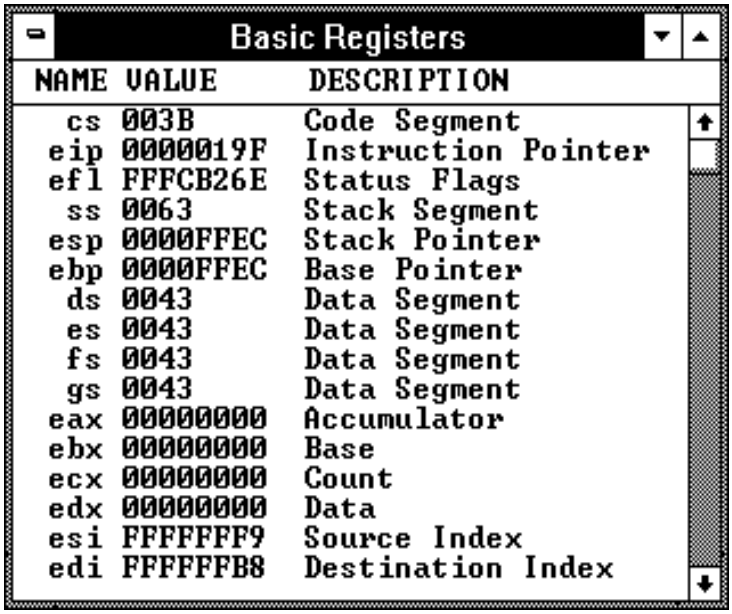
To display registers

- Choose the Window→Basic Registers command.

The register values displayed in the window are periodically updated to show you how the values change during program execution. The Status Flags register can be displayed and modified as decoded bits by double-clicking on its value.

When the register windows are updated, user program execution is temporarily interrupted. To prevent the user program from being interrupted (and the register windows 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 Basic Registers Window



To edit registers

1. Display the register contents by choosing the Window→Basic Registers command or the Window→System 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.

Tracing Program Execution

This section shows you how:

- To trace callers of a specified function
- To trace execution within a specified function
- To trace accesses to a specified variable
- To trace until the command is halted
- To stop a running trace
- To repeat the last 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 must satisfy a store-qualifier 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.

See Understanding 80386 Analysis to understand how the analyzer works with the prefetching, of the 80386, how the disassembler decodes the bus cycles, and how to use Execution Trace Messages to resolve questions about the exact target address of branches.

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 the state was captured (relative to the other states, by default).

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

Tracing Monitor Cycles

When the emulator is executing monitor code, cycles are generated, but usually not all cycles are captured by the analyzer. The 80386 emulator allows you to determine the types of monitor cycles to be captured when using the foreground monitor, and the types of monitor cycles to be ignored, as follows:

Default: By default, accesses by the monitor to non-monitor address space are traced. Execution of monitor code is not traced. This means that if the monitor reads memory to update the memory window, the trace list will show memory reads from the accessed addresses. However, the code executed by the monitor to read the addresses will not be shown.

Quiet: You can set up the emulator to prevent capture of states to target addresses while executing in the monitor. This will prevent the capture of monitor read cycles when the monitor updates the memory window, but it may also prevent capture of useful information, too. For example, if you are using the foreground monitor and an interrupt arrives, your interrupt code will execute but the analyzer will not capture its execution in the trace list.

Complete: You can set up the emulator to capture all states generated by the monitor. This will let you see the execution of the monitor in addition to its accesses to non-monitor address space. This is generally used to help debug a custom foreground monitor.

To set up the monitor trace options:

Default:

Settings→Extended→Trace Cycles→User

Settings→Emulator Config→Hardware... then make sure the "Enable Foreground Monitor Traced as User" box is checked.

Quiet:

Settings→Extended→Trace Cycles→User

Settings→Emulator Config→Hardware... then make sure the "Enable Foreground Monitor Traced as User" box is not checked.

Complete:

Settings→Extended→Trace Cycles→Both

Settings→Emulator Config→Hardware... then make sure the "Enable Foreground Monitor Traced as User" box is checked.

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.

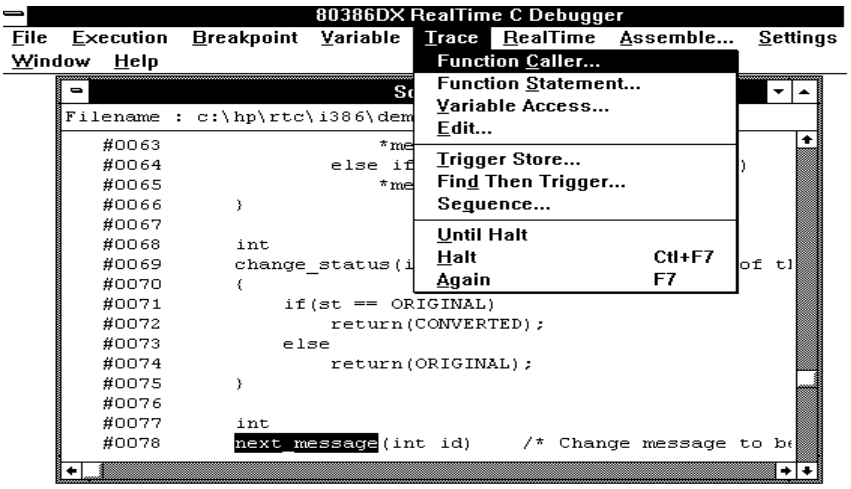
To identify interrupts in program execution, trace the caller of the interrupt process routine using the Trace→Function Caller... (ALT, T, C) command.

Example

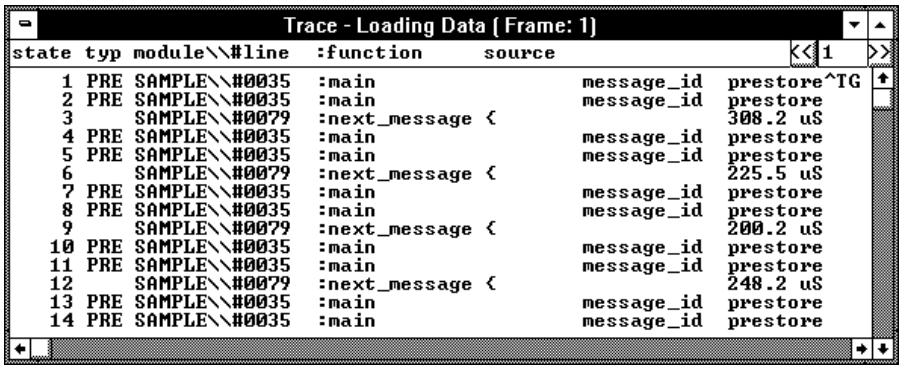
To trace the caller of "next_message":

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.

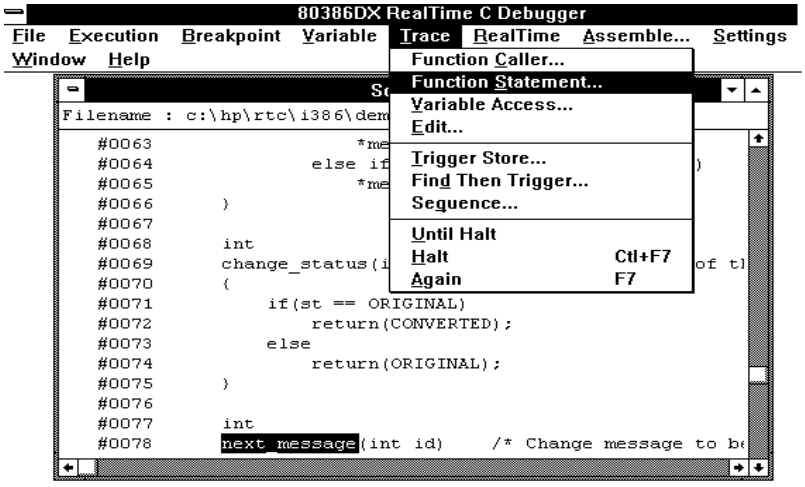
This command traces C functions only. It does not trace execution of assembly language subroutines.

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.

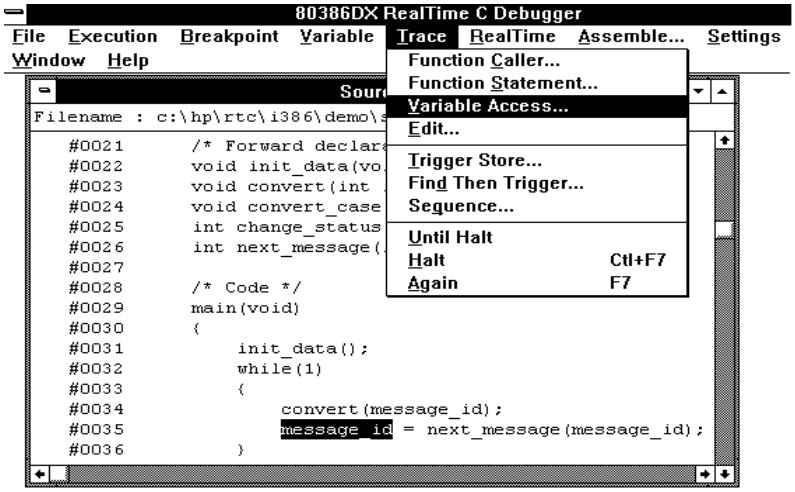
The command also traces access to the Assembler symbol specified by its name and size.

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 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:

  1. Stop the trace initiated with the Trace→Until Halt (ALT, T, U) command.
  2. Force termination of the trace that cannot be completed due to absence of the specified state.
  3. 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.

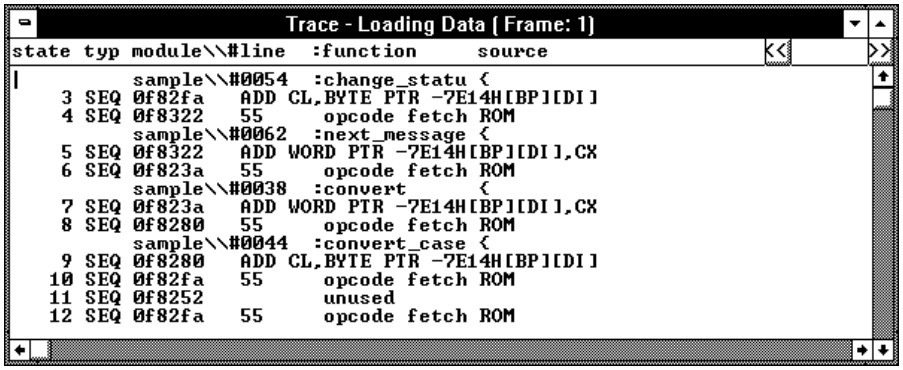
2 From the Trace window's control menu, choose the Display→Bus Cycle ON (ALT, -, D, B) command.

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.

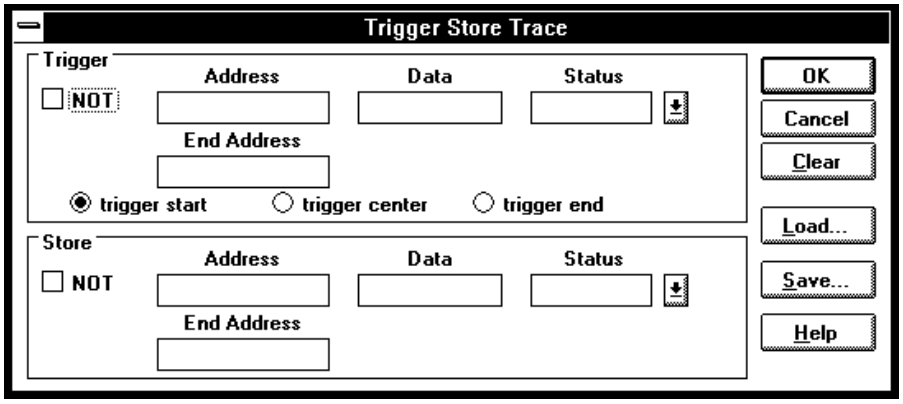
2 Specify the *trigger condition* using the Address, Data, and/or Status text boxes within the Trigger group box.

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:



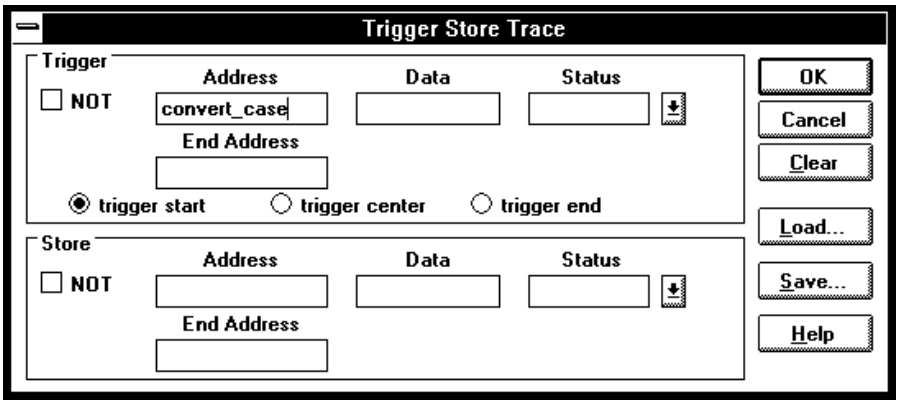
A group of Address, Data, and Status text boxes combine to form a *state qualifier*. You can specify an address range by entering a value in the End Address box. By selecting the NOT check box, you can specify all states other than those identified by the address, data, and *status values*.

Example

To trace execution after the "convert_case" function:

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

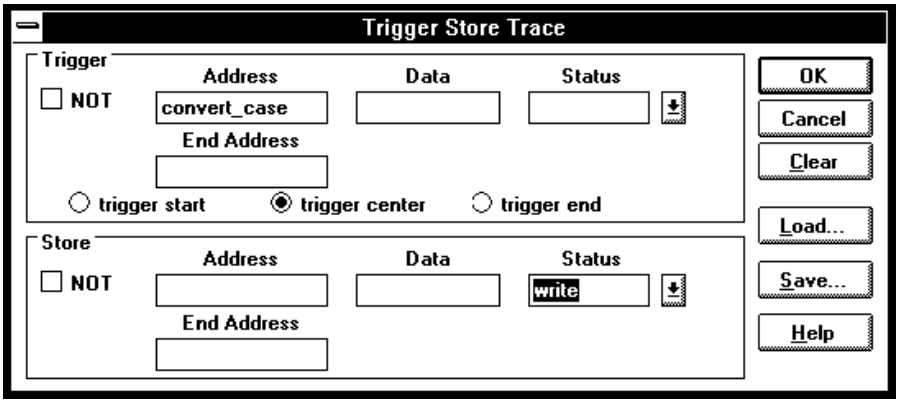
Enter "convert_case" in the Address text box in the Trigger group box.



Choose the OK button.

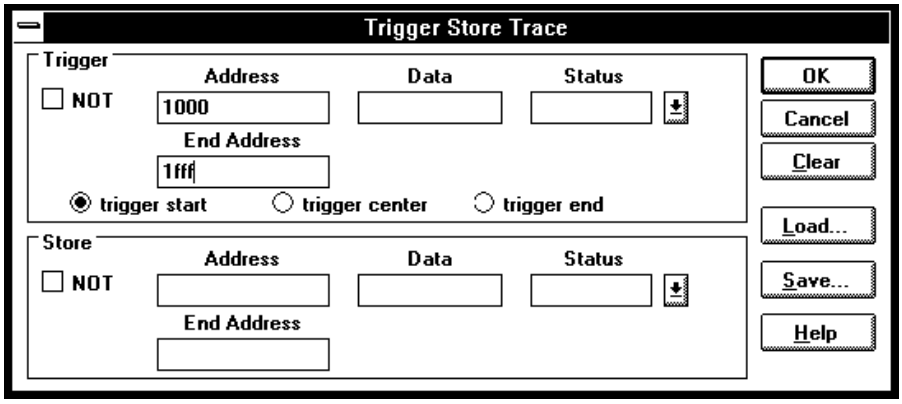
Example

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



Example

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



To set up a "Find Then Trigger" trace specification

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

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

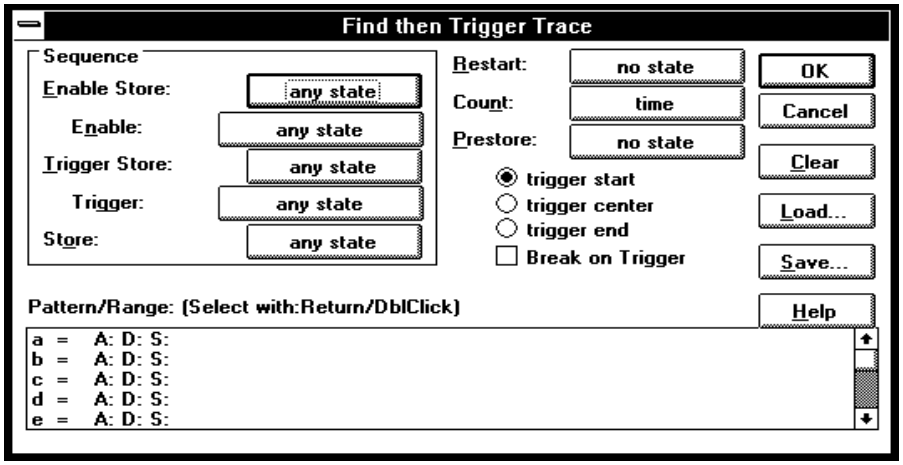
3 Specify the *restart*, *count*, and *prestore* conditions.

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

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

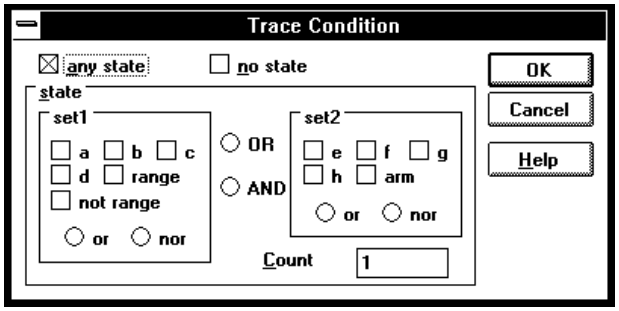
6 Choose the OK button to set up the analyzer and start the trace.

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



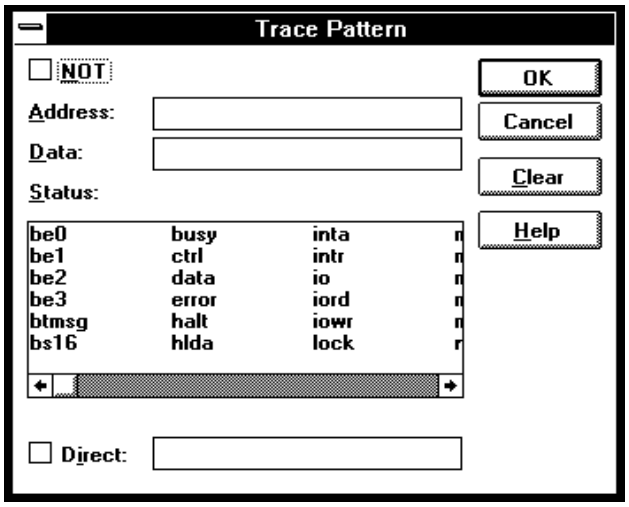
Choosing the enable, trigger, store, count, or prestore buttons opens a Condition dialog box that lets you select "any state," "no state," trace patterns

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

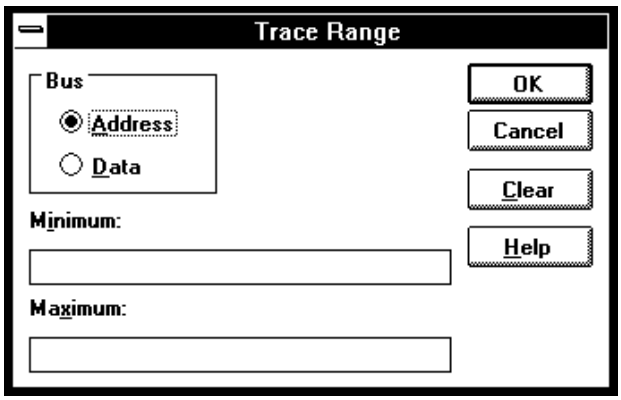


The range and pattern resources are defined by double-clicking on the resource name in the Pattern/Range list box.

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



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



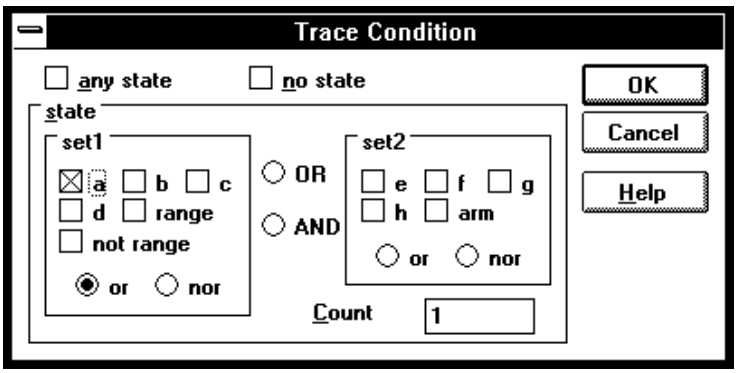
Example

To trace execution after the "convert_case" function:

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

Choose the Trigger button (default: any state).

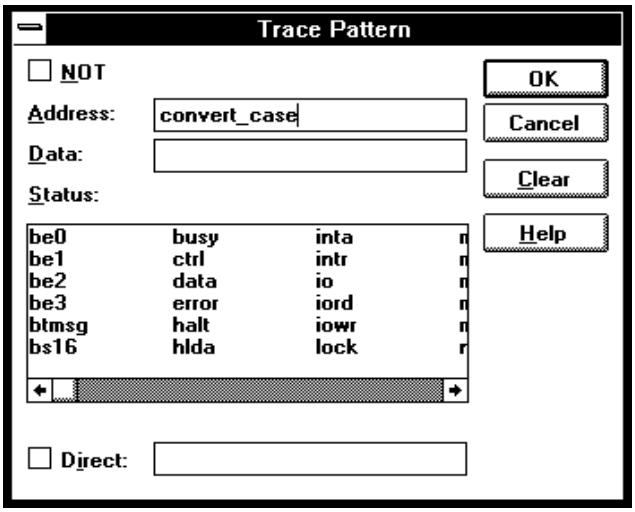
Select "a."



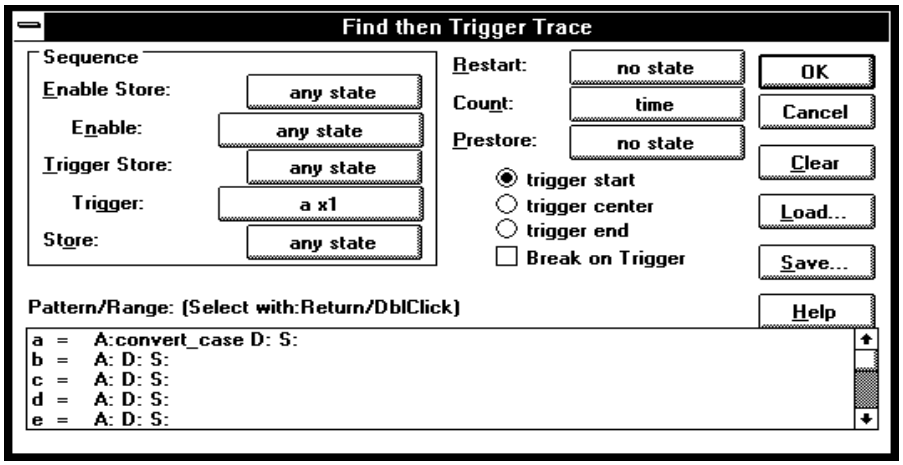
Choose the OK button.

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

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



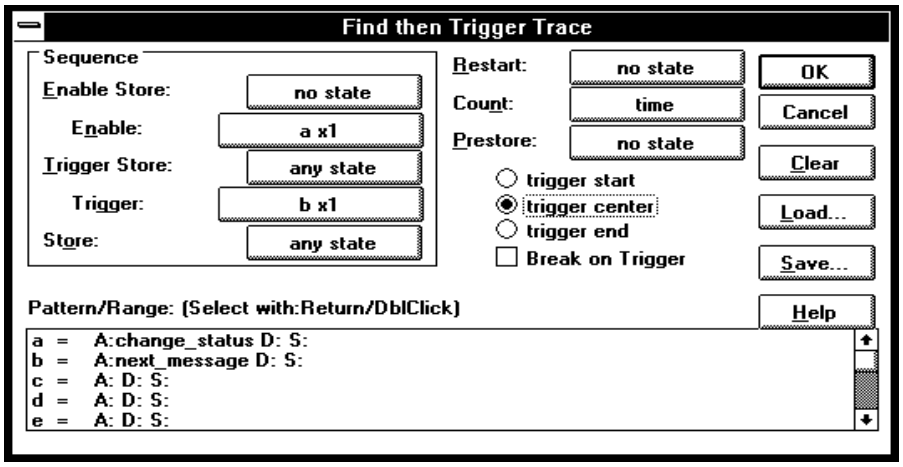
Choose the OK button in the Trace Pattern dialog box.



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

Example

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



To set up a "Sequence" trace specification

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

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

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

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

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

1 Choose the Trace→Sequence... (ALT, T, Q) command.

2 Specify the *primary branch*, *secondary branch*, and *store* conditions for each *sequence level* you will use.

3 Specify which sequence level to trigger on. The analyzer triggers on the entry to the specified level. Therefore, the condition that causes a branch to the specified level actually triggers the analyzer.

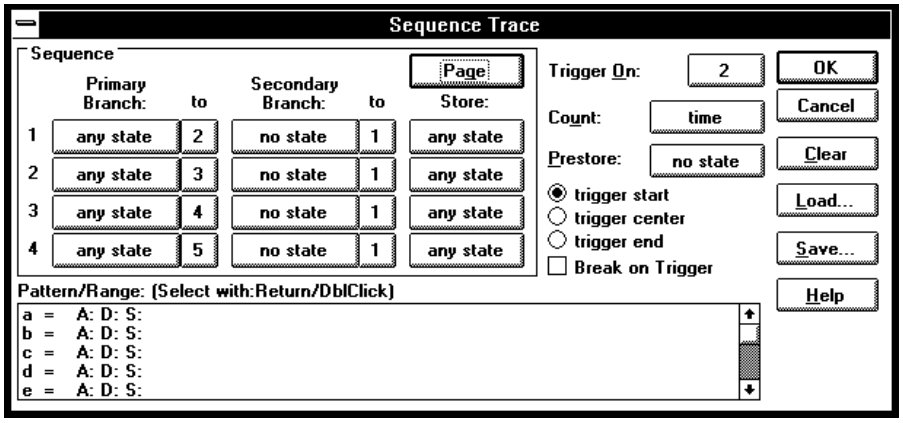
4 Specify the *count* and *prestore* conditions.

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

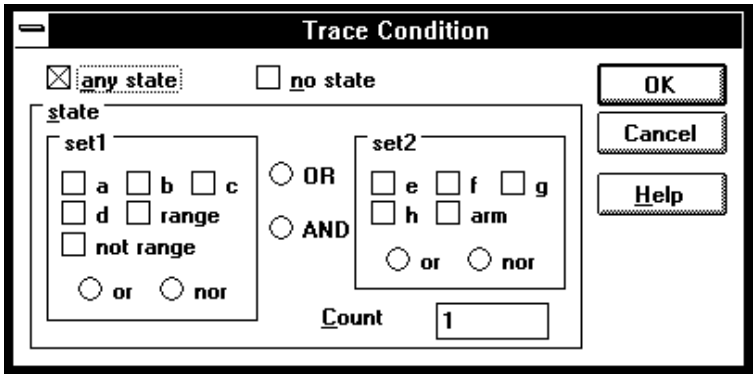
6 If you want emulator execution to break to the monitor when the trigger condition occurs, select the *Break On Trigger* check box.

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

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

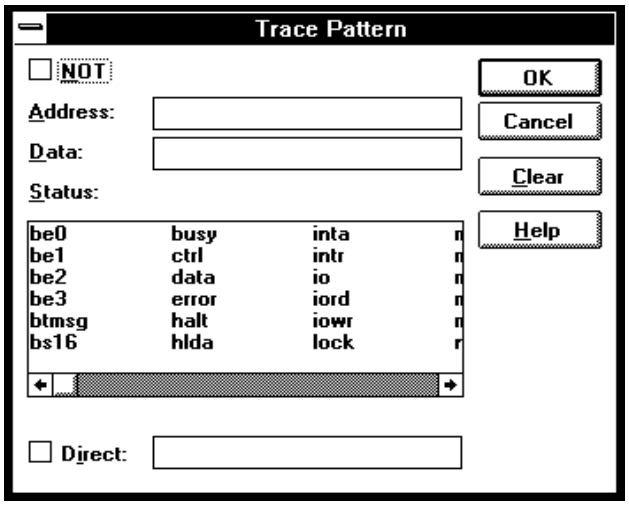


Choosing the primary branch, secondary branch, store, count, or prestore buttons opens a Condition dialog box that lets you select "any state," "no state," trace patterns "a" through "h," "range," or "arm" as the condition. Patterns "a" through "h," "range," and "arm" are grouped into two sets, and resources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

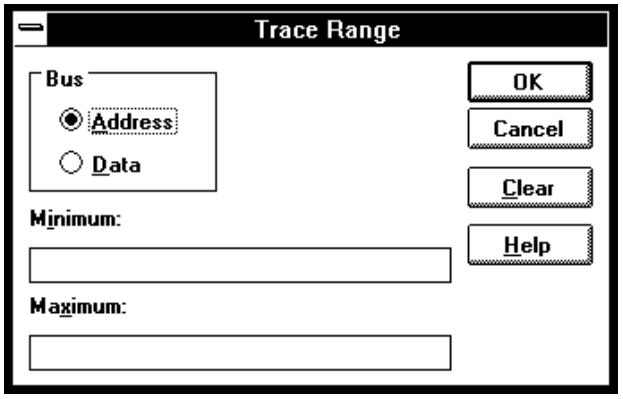


The range and pattern resources are defined by double-clicking on the resource name in the Pattern/Range list box.

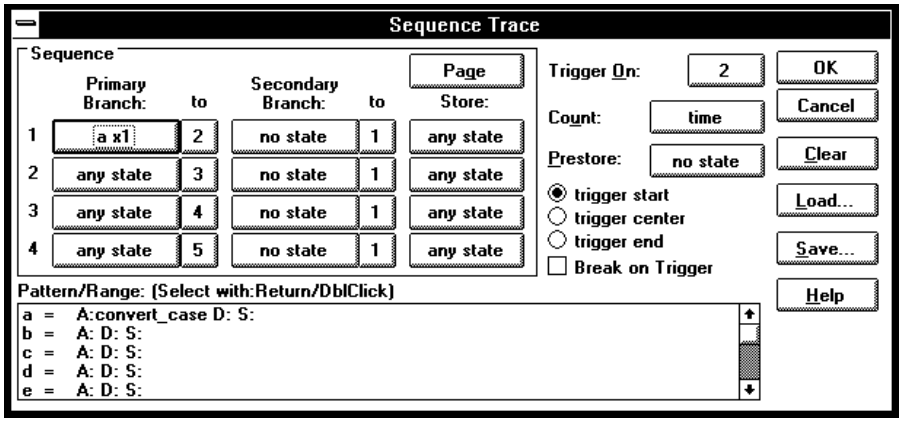
If you double-click on a pattern name, the Trace Pattern dialog box is opened to let you specify address, data, and status values. By selecting the NOT check box, you can specify all states other than those identified by the address, data, and *status values*. The Direct check box lets you specify status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



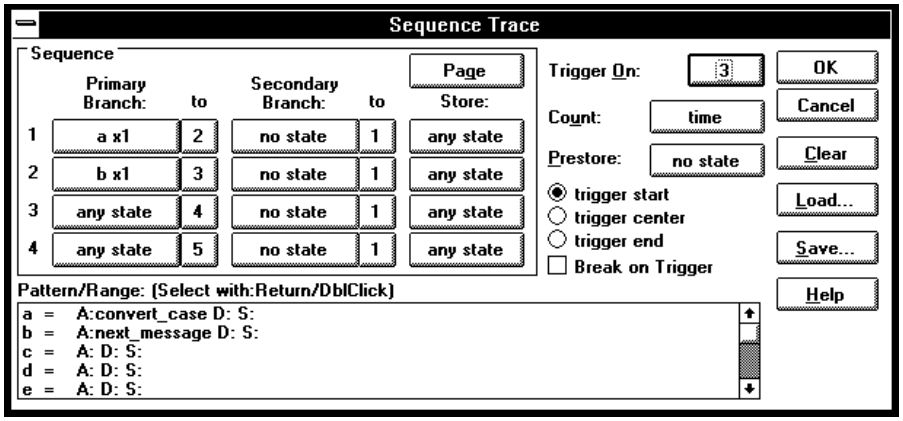
If you double-click on the range resource, the Trace Range dialog box is opened to let you select either the Address range option or the Data range option and enter the minimum and maximum values in the range.



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



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



To edit a trace specification

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

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

3 Choose the OK button.

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

To trace "windows" of program execution

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

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

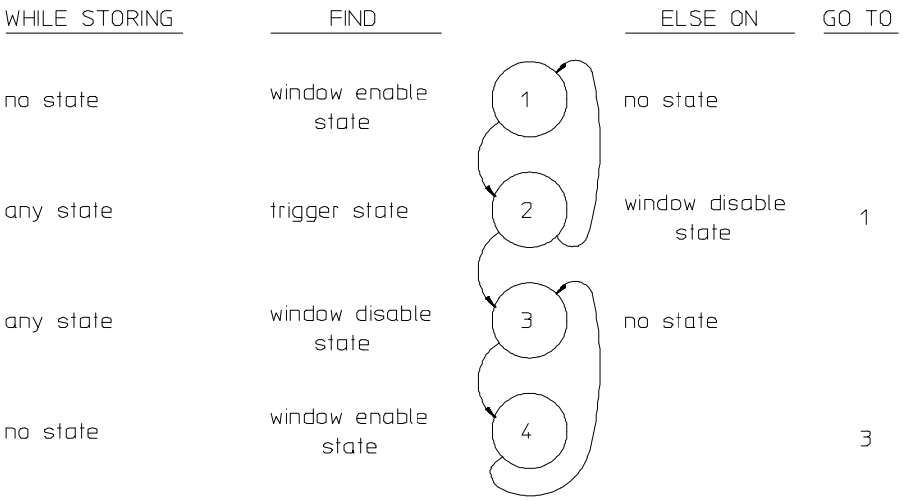
3 Choose the OK button.

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

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

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

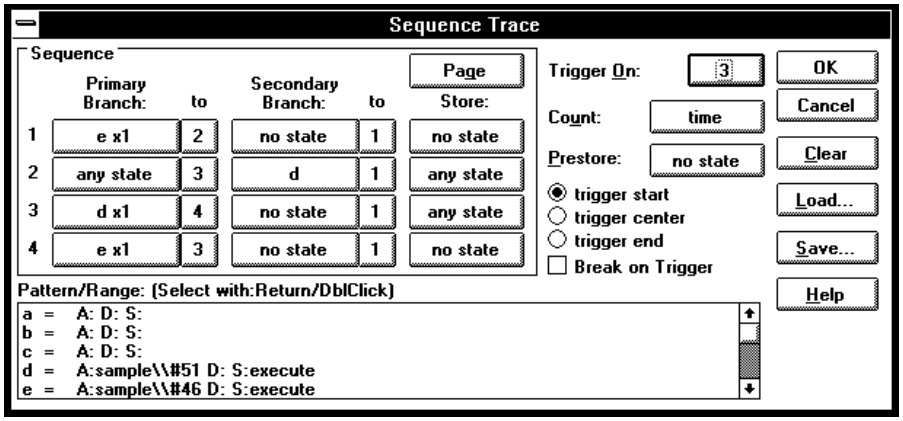
If you want to store the window of code execution before and after the trigger condition, use two sets of paired sequence levels: one window enable/disable pair of sequence levels before the trigger, and another disable/enable pair after the trigger as shown below.



Notice that the order of the second sequence level pair is swapped. 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 46 and 51 of the sample program, triggering on any state in the window:



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

To store the current trace specification

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

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

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

To load a stored trace specification

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

2 Choose the Load... button.

3 Select the desired trace specification file.

4 Choose the OK button.

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

Part 3

Reference

Descriptions of the product in a dictionary or encyclopedia format.

6

Command File and Macro Command Summary

Command File and Macro Command Summary

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

The characters in parentheses can be ignored for shortcut entry.

Run Control Commands

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

Variable and Memory Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
VARI(ABLE)	variable	TO	data		Changing value of variable
MEM(ORY)	FIL(L)	size	addr-range	value	Filling memory contents
MEM(ORY)	COP(Y)	size	addr-range	address	Copying memory contents
MEM(ORY)	IMA(GE)	size	addr-range		Copying target memory
MEM(ORY)	LOA(D)	MOT(OSREC)	file_name		Loading memory from a Motorola S-record file
MEM(ORY)	LOA(D)	INT(ELHEX)	file_name		Loading memory from an Intel Hexadecimal file
MEM(ORY)	STO(RE)	MOT(OSREC)	addr-range	file_name	Storing memory to a Mororola S-record file
MEM(ORY)	STO(RE)	INT(ELHEX)	addr-range	file_name	Storing memory to an Intel Hexadecimal file
MEM(ORY)	BYT(E)				Byte format display
MEM(ORY)	WOR(D)				16-Bit format display
MEM(ORY)	ABS(OLUTE)				Single-column display
MEM(ORY)	BLO(CK)				Multi-column display
MEM(ORY)	LON(G)				32-Bit format display
IO	BYTE/WORD	IOSPACE/MEMORY	address TO data		Editing specified I/O address
IO	SET	BYTE/WORD	IOSPACE/MEMORY	address	Registering I/O display
IO	DEL(ETE)	BYTE/WORD	IOSPACE/MEMORY	address	Deleting I/O address
WP	SET	address			Registering watchpoint

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

Breakpoint Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
BM	SET	address	command		Setting break macro
BM	SET	breakaddress	command		Setting break macro
BM	SET EXE(C)	breakaddress	command		Setting execution break macro
BM	SET ACC(ESS)	BYT(E)	breakaddress	command	Setting byte-access break macro
BM	SET ACC(ESS)	WOR(D)	breakaddress	command	Setting word-access break macro
BM	SET ACC(ESS)	DWO(RD)	breakaddress	command	Setting doubleword-access break macro
BM	SET WRI(TE)	BYT(E)	breakaddress	command	Setting byte-write break macro
BM	SET WRI(TE)	WOR(D)	breakaddress	command	Setting word-write break macro
BM	SET WRI(TE)	DWO(RD)	breakaddress	command	Setting doubleword-write break macro
BM	DEL(ETE)	address			Deleting break macro
BP	SET	address			Setting breakpoint
BP	SET EXE(C)	address			Setting execution breakpoint
BP	SET ACC(ESS)	BYT(E)	address		Setting byte-access breakpoint
BP	SET ACC(ESS)	WOR(D)	address		Setting word-access breakpoint
BP	SET ACC(ESS)	DWO(RD)	address		Setting doubleword-access breakpoint
BP	SET WRI(TE)	BYT(E)	address		Setting byte-write breakpoint
BP	SET WRI(TE)	WOR(D)	address		Setting word-write breakpoint
BP	SET WRI(TE)	DWO(RD)	address		Setting doubleword-write breakpoint
BP	DEL(ETE)	address			Deleting breakpoint
BP	DEL(ETE)	ALL			Deleting breakpoint
BP	DISABLE	address			Disabling a breakpoint
BP	ENABLE	address			Enabling a breakpoint
EVA(LUATE)	address				Expression window display
EVA(LUATE)	"strings"				Printing string
EVA(LUATE)	CLE(AR)				Clearing Expression window

Window Open/Close Command

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

Configuration Command

Command	Param_1	Param_2	Param_3	Param_4	Operation
MON(ITOR)	TYPE	FOREGROUND			Selects foreground monitor
MON(ITOR)	TYPE	BACKGROUND			Selects background monitor
MON(ITOR)	SELECTOR	selector			Specifies monitor's selector
MON(ITOR)	TRDY	ENA(BLE)			Interlock with target RDY
MON(ITOR)	TRDY	DISABLE			Ignore target system RDY
MON(ITOR)	FIL(ENAME)	NONE			Use built-in foreground monitor
MON(ITOR)	FIL(ENAME)	file_name			Use named monitor object file
CON(FIG)	STA(RT)				Starting configuration
CON(FIG)	config-item	config-ans			Executing configuration
CON(FIG)	END				Ending configuration
MAP	STA(RT)				Starting mapping
MAP	addr-range	mem_type	attributes		Executing mapping
MAP	OTHER	mem_type			Mapping OTHER area
MAP	END				Ending mapping
ADDRTRAN	STA(RT)				Starting address translation
ADDRTRAN	config-item	config-ans			Executing address translation
ADDRTRAN	END				Ending address translation
MOD(E)	MNE(MONIC)	ON			Enabling Mnemonic display
MOD(E)	MNE(MONIC)	OFF			Enabling Source display
MOD(E)	REA(LTIME)	ON			Enabling real-time mode
MOD(E)	REA(LTIME)	OFF			Disabling real-time mode
MOD(E)	IOG(UARD)	ON			Enabling I/O guard
MOD(E)	IOG(UARD)	OFF			Disabling I/O guard
MOD(E)	DOW(NLOAD)	NOE(RRABORT)			Load file or memory; ignore errors
MOD(E)	DOW(NLOAD)	ERR(ABORT)			Load file or memory; abort if error
MOD(E)	MEM(ORYPOLL)	ON			Enabling Memory polling
MOD(E)	MEM(ORYPOLL)	OFF			Disabling Memory polling
MOD(E)	WAT(CHPOLL)	ON			Enabling WatchPoint polling
MOD(E)	WAT(CHPOLL)	OFF			Disabling WatchPoint polling
MOD(E)	LOG	ON			Enabling log file output
MOD(E)	LOG	OFF			Disabling log file output
MOD(E)	BNC	INP(UT_ARM)			Setting BNC input
MOD(E)	BNC	OUT(PUT_TRIGGER)			Setting BNC output
MOD(E)	SYM(BOLCASE)	ON			Case sensitive symbol search
MOD(E)	SYM(BOLCASE)	OFF			Case insensitive sym. search
MOD(E)	TRACECLOCK	BACKGROUND			Trace background cycles
MOD(E)	TRACECLOCK	BOTH			Trace all processor cycles
MOD(E)	TRACECLOCK	USER			Trace user program cycles
MOD(E)	TRACE	DISPLAY	FROM	<state>	Trace disassembly begin from <state>
MOD(E)	TRACE	DISPLAY	BYTE0/1/2/3		Trace disassembly begin from BYTE<no.>
MOD(E)	TRACE	DISPLAY	USE16/USE32		Trace disassembly from 16-bit/32-bit segment type
MOD(E)	SOU(RCE)	ASK(PATH)			Prompt for source paths
MOD(E)	SOU(RCE)	NOA(SKPATH)			Don't prompt for source paths

File Command

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

Trace Commands

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

Symbol Window Commands

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

Command File Control Command

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

Global/Local Descriptor Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
GDT	SELECTOR	value			Obtain value of GDT selector
LDT	SELECTOR	value			Obtain value of LDT selector
GDT	ENTRY	value			Obtain value of GDT entry
GDT	ENTRY	value			Obtain value of LDT entry

Miscellaneous Commands

Command	Param_1	Param_2	Param_3	Param_4	Operation
ASM	address	label	"inst_string"		In-line assembler
BEE(P)					Sounding beep
BUTTON	label	"command"			Adds button to Button window
QUI(T)					Exiting debugger
QUI(T)	LOC(KED)				Exiting debugger while retaining control
COP(Y)	TO	file_name			Specifying copy destination
COP(Y)	SOU(RCE)				Copying Source window
COP(Y)	REG(ISTER)				Copying Register window
COP(Y)	MEM(ORY)				Copying Memory window
COP(Y)	WAT(CHPOINT)				Copying WatchPoint window
COP(Y)	BAC(KTRACE)				Copying BackTrace window
COP(Y)	IO				Copying I/O window
COP(Y)	EXP(RESSION)				Copying Expression window
COP(Y)	BUT(TON)				Copying Button window
CUR(SOR)	address				Positioning cursor
NOP					Non-operative
NOP	comments				Non-operative to prefix comment lines
SEA(RCH)	STR(ING)	FOR/BACK	ON/OFF	strings	Searching string
SEA(RCH)	FUN(CTION)	func_name			Selecting function
SEA(RCH)	MEM(ORY)	BYTE/WORD/LONG	addr-range	value	Searching memory
SEA(RCH)	MEM(ORY)	STR(ING)	"strings"		Searching memory for string
TER(MCOM)	ti-command				Terminal Interface command

Parameters

Parameter	Description	Notation
address	Address	See "Reference."
addr-range	Address range	
args	Arguments	Replaces placeholders in command file.
attributes		Can be comma-separated dp=dual-port mem;trdy=target RDY
breakaddress	linenumber, plinenum, or address.	See descriptions in this list.
case	Case sensing	
command	Macro command	Commands listed in the "Reference."
config-ans	Setting	See "Reference."
config-item	Configuration	See "Reference."
count	Count	Decimal notation
direction	Search direction	
directoryname	Directory name	
file_name	File name	
format	Memory file format	
func_name	Function name	
label	Button label	
linenumber	Line number	
mem_type	Memory type	
module_name	Module name	
mon-ans	Setting	See "Reference."
mon-item	Configuration	See "Reference."
plinenum	Macro line number	line number.macro number (ex. 34.1)
regname	Register name	
seconds	Time in seconds	
size	Data size	
space	Memory or I/O space	
strings	String "string"	
symbol_nam	Symbol name	
usersymbol	User-defined symbol	See "Reference."
value	Value	See "Reference."
window-name	Name of window	See "Reference."

WAIT Command Dialog Box

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

Choosing the STOP button cancels the WAIT command.

7

Expressions in Commands

Expressions in Commands

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

- Numeric constants (hexadecimal, decimal, octal, or binary values). You can only use Numeric constants when using the constant-address syntax.
- Symbols (identifiers).
- C operators (pointers, arrays, structures, unions, unary minus operators) and parentheses (specifying the order of operator evaluation).

Numeric Constants

All numeric constants are assumed to be hexadecimal, except when the number refers to a count; count values are assumed to be decimal.

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

Hexadecimal	Alphanumeric strings starting with "0x" or "0X" and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 0x12345678, 0xFFFF0000).
	Alphanumeric strings starting with any of '0' through '9', ending with 'H' or 'h', and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 12345678H, 0FFFF0000h).
	Alphanumeric strings starting with any of '0' through '9' and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 12345678, 0FFFF0000).
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
```

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

```
sample\\#22
```

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

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

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

```
dat→message,s
```

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

```
dat→message,x32
```

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

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

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

```
&data[0]
```

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

```
*1000
```

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

C Operators

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

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

8

Menu Bar Commands

Menu Bar Commands

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

- File→Load Object... (ALT, F, L)
- File→Command Log→Log File Name... (ALT, F, C, N)
- File→Command Log→Logging ON (ALT, F, C, O)
- File→Command Log→Logging OFF (ALT, F, C, F)
- File→Run Cmd File... (ALT, F, R)
- File→Load Debug... (ALT, F, D)
- File→Save Debug... (ALT, F, S)
- File→Load Emulator Config... (ALT, F, E)
- File→Save Emulator Config... (ALT, F, V)
- File→Copy Destination... (ALT, F, P)
- File→Exit (ALT, F, X)
- File→Exit HW Locked (ALT, F, H)
- Execution→Run (ALT, E, U)
- Execution→Run to Cursor (ALT, R C)
- Execution→Run to Caller (ALT, E, T)
- Execution→Run... (ALT, E, R)
- Execution→Single Step (ALT, E, N)
- Execution→Step Over (ALT, E, O)
- Execution→Step... (ALT, E, S)
- Execution→Break (ALT, E, B)

- Execution→Reset (ALT, E, E)
- Breakpoint→Set at Cursor (ALT, B, S)
- Breakpoint→Delete at Cursor (ALT, B, D)
- Breakpoint→Set Macro... (ALT, B, M)
- Breakpoint→Delete Macro (ALT, B, L)
- Breakpoint→Edit... (ALT, B, E)
- Variable→Edit... (ALT, V, E)
- Trace→Function Caller... (ALT, T, C)
- Trace→Function Statement... (ALT, T, S)
- Trace→Variable Access... (ALT, T, V)
- Trace→Edit... (ALT, T, E)
- Trace→Trigger Store... (ALT, T, T)
- Trace→Find Then Trigger... (ALT, T, D)
- Trace→Sequence... (ALT, T, Q)
- Trace→Until Halt (ALT, T, U)
- Trace→Halt (ALT, T, H)
- Trace→Again (ALT, T, A)
- RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D)
- RealTime→Monitor Intrusion→Allowed (ALT, R, T, A)
- RealTime→I/O Polling→ON (ALT, R, I, O)
- RealTime→I/O Polling→OFF (ALT, R, I, F)
- RealTime→Watchpoint Polling→ON (ALT, R, W, O)
- RealTime→Watchpoint Polling→OFF (ALT, R, W, F)
- RealTime→Memory Polling→ON (ALT, R, M, O)

- RealTime→Memory Polling→OFF (ALT, R, M, F)
- Assemble... (ALT, A)
- Settings→Emulator Config→Hardware... (ALT, S, E, H)
- Settings→Emulator Config→Memory Map... (ALT, S, E, M)
- Settings→Emulator Config→Monitor... (ALT, S, E, O)
- Settings→Emulator Config→Address Translation... (ALT, S, E, A)
- Settings→Communication... (ALT, S, C)
- Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O)
- Settings→BNC→Input to Analyzer Arm (ALT, S, B, I)
- Settings→Font... (ALT, S, F)
- Settings→Tabstops... (ALT, S, T)
- Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O)
- Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F)
- Settings→Extended 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)
- 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.

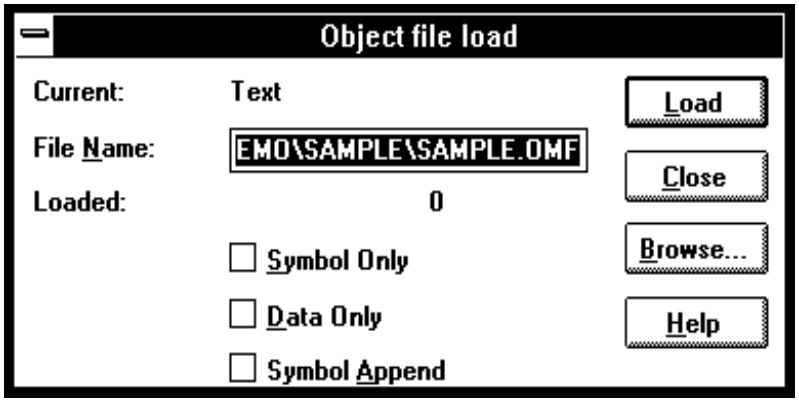
Program code is loaded into emulation memory or target system RAM.

Object files are typically Intel OMF386 boot-loadable format absolute files.

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

Load Object File Dialog Box

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



Current	Shows the currently loaded object file.
File Name	Specifies the object file to be loaded.
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 if the load was successful. The dialog box is left open on screen if the load was not successful.
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.

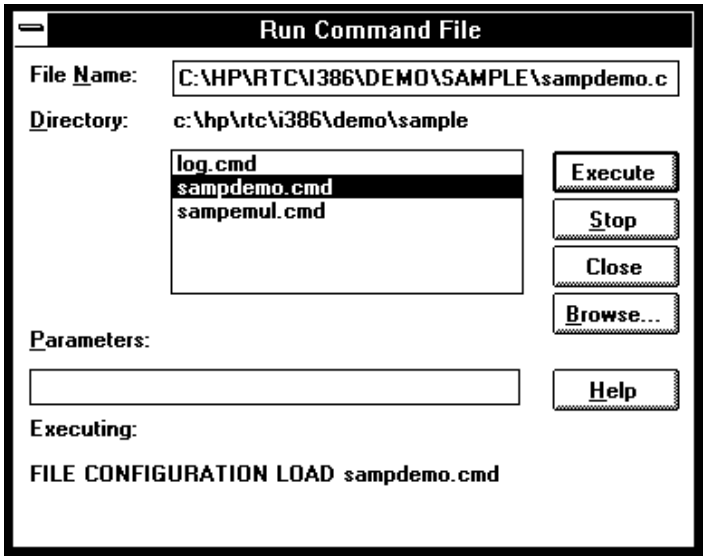
Command files can be:

- 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:



File Name — Lets you enter the name of the command file to be executed.

Directory	Shows the current directory and the command files in that directory. You can select the command file name from this list.
Parameters	Lets you specify up to five parameters that replace placeholders $1 through $5 in the command file. Parameters must be separated by blank spaces.
Executing	Shows the command being executed.
Execute	Executes the command file.
Stop	Stops command file execution.
Close	Closes the dialog box.
Browse...	Opens a file selection dialog box from which you can select the command file name.

Command File Command

```
FIL(E) COM(MAND) filename args
```

See Also

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

File→Load Debug... (ALT, F, D)

Loads a debug environment file.

This command opens a file selection dialog box from which you select the debug environment file.

Debug environment files have the extension ".ENV".

Debug environment files contain information about:

- Breakpoints.
- Variables in the WatchPoint window.
- The directory that contains the currently loaded object file.

Command File Command

```
FIL(E) ENV(IRONMENT) LOA(D) filename
```

File→Save Debug... (ALT, F, S)

Saves a debug environment file.

This command opens a file selection dialog box from which you select the debug environment file.

The following information is saved in the debug environment file:

- Breakpoints.
- Variables in the WatchPoint window.
- The directory that contains the currently loaded object file.

Command File Command

```
FIL(E) ENV(IRONMENT) SAV(E) filename
```

File→Load Emulator Config... (ALT, F, E)

Loads a hardware configuration command file.

This command opens a file selection dialog box from which you select the hardware configuration file.

Emulator configuration command files contain:

- Hardware configuration settings.
- Memory map configuration settings.
- Monitor configuration settings.

Command File Command

```
FIL(E) CON(FIGURATION) LOA(D) filename
```

See Also

"To load an emulator configuration" in the "Saving and Loading Configurations" section of the "Configuring the Emulator" chapter.

File→Save Emulator Config... (ALT, F, V)

Saves the current hardware configuration to a command file.

The following information is saved in the emulator configuration file:

- Hardware configuration settings.
- Memory map configuration settings.
- Monitor configuration settings.

Command File Command

```
FIL(E) CON(FIGURATION) STO(RE) filename
```

See Also

"To save the current emulator configuration" in the "Saving and Loading Configurations" section of the "Configuring the Emulator" chapter.

File→Copy Destination... (ALT, F, P)

Names the listing file to which debugger information may be copied.

The contents of most of the debugger windows can be copied to the destination listing file by choosing the Copy→Window command from the window's control menu.

The Symbol and Trace windows' control menus provide the Copy→All command for copying all of the symbolic or trace information to the destination listing file.

This command opens a file selection dialog box from which you select the name of the output list file. Output list files have the extension ".LST".

Command File Command

```
COP(Y) TO filename
```

See Also

"To change the list file destination" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

File→Exit (ALT, F, X)

Exits the debugger.

Command File Command

```
QUI(T)
```

See Also

"To exit the debugger" in the "Starting and Exiting the Debugger" section of the "Using the Debugger Interface" chapter.

File→Exit HW Locked (ALT, F, H)

File→Exit HW Locked (ALT, F, H)

Exits the debugger and locks the emulator hardware.

When the emulator hardware is locked, your user name and ID are saved in the HP 64700 and other users are prevented from accessing it.

You can restart the debugger and resume your debug session after reloading the symbolic information with the File→Load Object... (ALT, F, L) command.

Command File Command

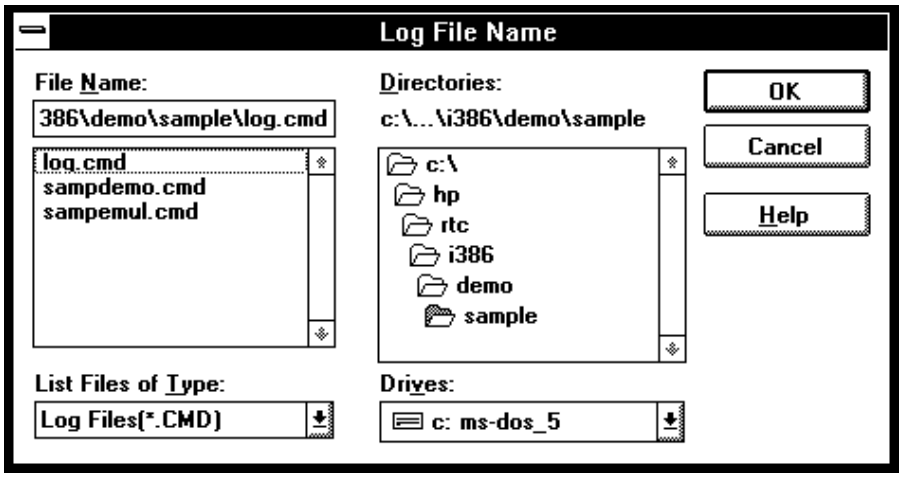
```
QUI(T) LOC(KED)
```

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 you already have four hardware breakpoints.

If the cursor-selected source line is not reached within the number of milliseconds specified by StepTimerLen in the B3637.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.

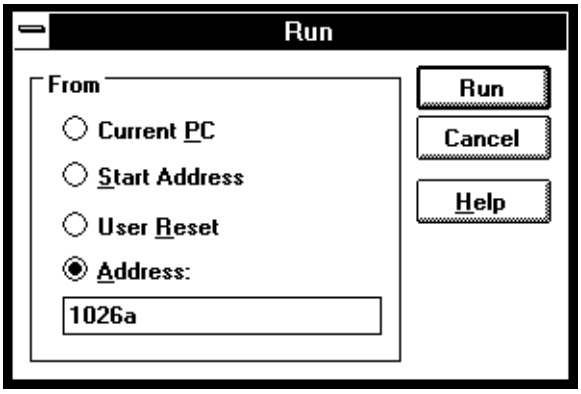
This command sets the processor status to RUNNING IN USER PROGRAM.

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 resets the processor (driving the "flying lead" low); then releases reset, causing the processor to begin executing at the reset address (0ffffff0).
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.

During a single-step command, multiple instructions can be executed if the instruction being stepped causes an instruction fault or task switch.

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)

"Unexpected Stepping Behavior" in the "Concepts" chapter.

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.

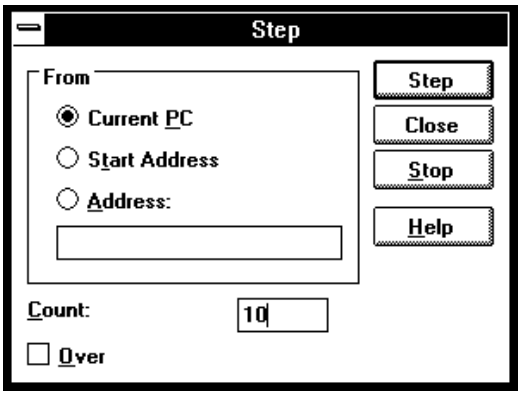
When in the mnemonic mixed display mode, single assembly language instructions are executed.

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.
Step	Single-steps the specified number of instructions or source lines, starting from the specified address.
Close	Closes the dialog box.
Stop	Stops single-stepping.

Command File Command

```
STE(P) count
Single-steps the specified number of instructions or
source lines, starting from the current program counter
address.

STE(P) count address
Single-steps the specified number of instructions or
source lines, starting from the specified address.

STE(P) count STA(RT)
Single-steps the specified number of instructions or
source lines, starting from the transfer address
defined in the object file.

OVE(R) count
Single-steps the specified number of instructions or
source lines, starting from the current program counter
address.  If an instruction or source line makes a
subroutine or function call, the entire subroutine or
function is executed.

OVE(R) count address
Single-steps the specified number of instructions or
source lines, starting from the specified address. If
an instruction or source line makes a subroutine or
function call, the entire subroutine or function is
executed.

OVE(R) count STA(RT)
Single-steps the specified number of instructions or
source lines, starting from the transfer address
defined in the object file.  If an instruction or
source line makes a subroutine or function call, the
entire subroutine or function is executed.
```

See Also

"To step multiple lines or instructions" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Single Step (ALT, E, N)
Execution→Step Over (ALT, E, O)

Execution→Break (F4), (ALT, E, B)

Stop user program execution and break into the monitor.

This command can also be used to break into the monitor when the processor is in the EMULATION RESET status.

Once the command has been completed, the processor transfers to the RUNNING IN MONITOR status.

Command File Command

BRE(AK)

See Also

"To stop program execution" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Reset (ALT, E, E)

Resets the emulation microprocessor.

If a foreground monitor is being used, it will automatically be loaded when this command is chosen.

While the processor is in the EMULATION RESET state, no display or modification is allowed for the contents of target system memory or registers. Therefore, before you can display or modify target system memory or processor registers, you must use the Execution→Break (ALT, E, B) command to break into the monitor.

Note

If RealTime→Monitor Intrusion→Allowed is selected, the emulation microprocessor may switch immediately from reset to running in 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.

There are two types of breakpoints available: software and hardware

Software breakpoints

Software breakpoints are handled by the 80386DX bond-out's interrupt facility. When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with the bond-out's breakpoint interrupt instruction (which is different than INT 3).

Note that you must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data).

Also, in order to successfully set a software breakpoint, the emulator must be able to write to the memory location specified. Therefore, software breakpoints cannot be set in target ROM. If the emulator discovers an attempt to put a software breakpoint in target ROM, it will automatically attempt to use a hardware breakpoint. If you already have four hardware breakpoints, this will fail. You can, however, copy a target ROM memory image into emulation memory, then use a software breakpoint.

Hardware breakpoints

Hardware breakpoints use the 80386 bond-out's breakpoint facility. It shares the debug 'breakpoint' registers with the breakpoint registers available to the target system, so when hardware breakpoint registers are used by the emulator, they are unavailable for use by the target system's software. Any attempt by the target system software to use the hardware breakpoint will result in a break to the monitor.

There are four hardware breakpoints for the 80386.

Hardware breakpoints are used automatically when the emulator attempts to set a breakpoint and detects that the memory value did not change (probably because it is in ROM).

The Breakpoint→Set at Cursor (ALT, B, S) command may cause BP markers to appear at two or more addresses. This happens when a single instruction is associated with two or more source lines. You can select the mnemonic display mode to verify that the breakpoint is set at a single address.

Command File Command

```
BP SET address
```

See Also

"To set a breakpoint" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

Breakpoint→Delete at Cursor (ALT, B, D)

Deletes the breakpoint set at the cursor-selected address in the Source window.

This command is only applicable to lines that contain "BP" markers (which indicate set breakpoints). Once the breakpoint is deleted, the original instruction is replaced.

Command File Command

```
BP DEL(ETE) address
```

See Also

"To delete a single breakpoint" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

Breakpoint→Edit... (ALT, B, E)

Breakpoint→Set Macro... (ALT, B, M)

Sets a *break macro* immediately before the cursor-selected address in the Source window.

Break macro lines are marked with the "BP" breakpoint marker, and the corresponding addresses or line numbers are displayed in decimal format.

When a break macro is hit, program execution stops immediately before executing the instruction or source code line at which the break macro is set. Then, the commands associated with the break macro are executed. When a "RUN" command is set as the last command in the break macro, the system executes the break macro and resumes program execution.

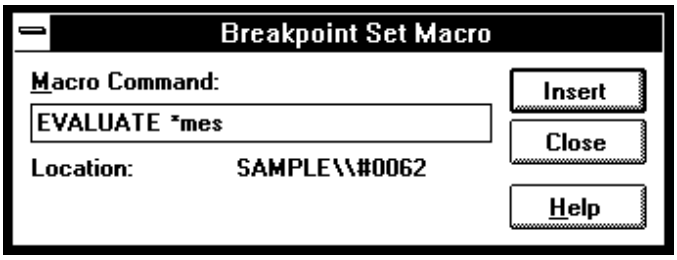
The break macro remains active until it is deleted with the Breakpoint→Delete Macro (ALT, B, L) command or the Breakpoint→Edit... (ALT, B, E) command.

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:



Macro Command	Specifies the command to be added to the break macro.
Location	Displays the specified line number or address followed by a decimal point and the break macro line number.
Insert	Inserts the specified macro command at the location immediately preceding the specified source line or address, or inserts the macro command at the location immediately following the specified break macro line. Two or more commands can be associated with a break macro by entering the first command and choosing Insert, then entering the second command and choosing Insert, and so on. Commands execute in the order of their entry.
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.

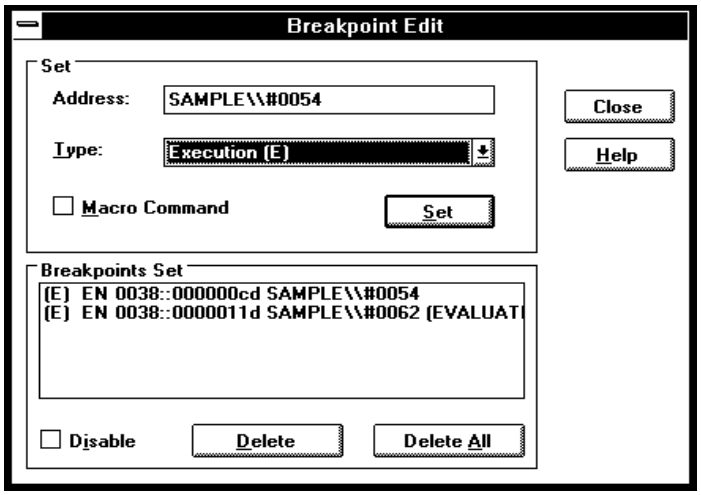
Breakpoint→Edit... (ALT, B, E)

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:



Address — Lets you specify the address at which to set a breakpoint or a break macro.

Type — Allows you to choose the type of breakpoint to cause a break into the monitor: (Note that the 80386 has only four hardware breakpoint registers.)

Execution (E). A break occurs when the opcode at the address is about to be executed. A software breakpoint is used unless the address is in target ROM. In that case, a hardware breakpoint register is used in the 80386 processor

Execution Hardware Only (EH). A break occurs when the opcode at the address is about to be executed. Only hardware breakpoints are used (that is, one of the four hardware breakpoint registers in the 80386 is used to implement the breakpoint).

Write Byte (WB). A break occurs when the byte specified by the address is written to. This is implemented by using one of the hardware breakpoint registers in the 80386 processor.

Write Word (WW). A break occurs when the word (16 bits) specified by the address is written to. This is implemented by using one of the hardware breakpoint registers in the 80386 processor.

Write Dword (WD). A break occurs when the double word (32 bits) specified by the address is written to. This is implemented by using one of the hardware breakpoint registers in the 80386 processor.

Read/Write Byte (RB). A break occurs when the byte specified by the address is read from or written to. This is implemented by using one of the hardware breakpoint registers in the 80386 processor.

Read/Write Word (RW). A break occurs when the word (16 bits) specified by the address is read from or written to. This is implemented by using one of the hardware breakpoint registers in the 80386 processor.

Read/Write Dword (RD). A break occurs when the double word (32 bits) specified by the address is read from or written to. This is implemented by using one of the hardware breakpoint registers in the 80386 processor.

Macro command	When selected, this specifies that a break macro should be included with the breakpoint.
Set	Sets a breakpoint or a break macro at the specified address.

Breakpoints Set	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. One of the following may precede the name of the breakpoint in the "Breakpoints Set" text box: If EN precedes the breakpoint, the breakpoint is currently enabled. If DI precedes the breakpoint, the breakpoint is disabled. That is, it is not actually inserted into the code (or the hardware register is not enabled. See below). If two dashes (--) precede the breakpoint, the status is unknown (probably because you used the "Realtime→Monitor Intrusion→Disallowed" command).
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.
Disable	Disables the selected breakpoint.
Close	Closes the dialog box.

Note

Whenever a file is loaded (via the "File→Load Object" command), all breakpoints will be deleted. If you want to save your breakpoints, use the "File→Save Debug..." command.

Command File Command

```
BP SET address

BP DEL(ETE) ALL
```

```
BP DEL(ETE) address

BP ENA(BLE) address

BP DIS(ABLE) address

BP SET EXE(C) address

BP SET ACC(ESS) BYT(E) address

BP SET ACC(ESS) WOR(D) address

BP SET ACC(ESS) DWO(RD) address

BP SET WRI(TE) BYT(E) address

BP SET WRI(TE) WOR(D) address

BP SET WRI(TE) DWO(RD) address

BM SET breakaddress command

BM SET EXE(C) breakaddress command

BM SET ACC(ESS) BYT(E) breakaddress command

BM SET ACC(ESS) WOR(D) breakaddress command

BM SET ACC(ESS) DWO(RD) breakaddress command

BM SET WRI(TE) BYT(E) breakaddress command

BM SET WRI(TE) WOR(D) breakaddress command

BM SET WRI(TE) DWO(RD) breakaddress command
```

See Also

"To enable or disable software breakpoints" to understand how to enable breakpoints and the side-effects of doing so, in the "Setting the Hardware Options" section of the "Configuring the Emulator" chapter.

"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.

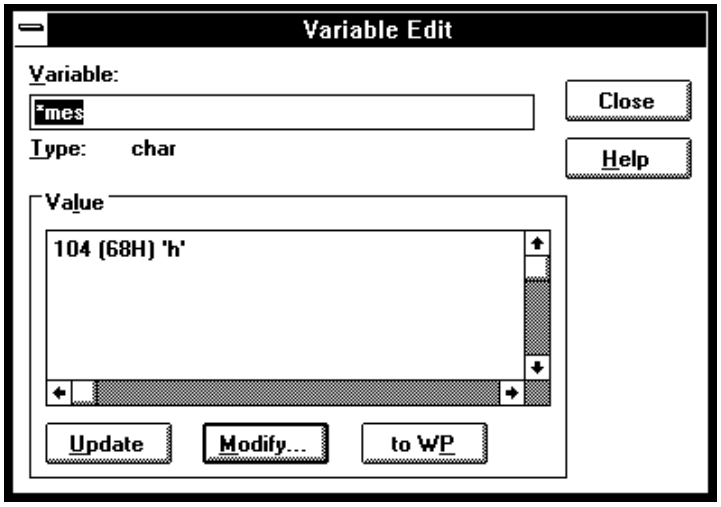
Variable→Edit... (ALT, V, E)

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 another window, automatically appears in this text box.
Type	Displays the type of the specified variable.
Value	Displays the contents of the specified variable.

Update	Reads and displays the contents of the variable specified in the Variable text box.
Modify	Modifies the contents of the specified variable. Choosing this button opens the Variable Modify Dialog Box, which lets you edit the contents of the variable.
to WP	Adds the specified variable to the WatchPoint window.
Close	Closes the dialog box.

Command File Command

```
VARI(ABLE) variable TO data
Replaces the contents of the specified variable with
the specified value.
```

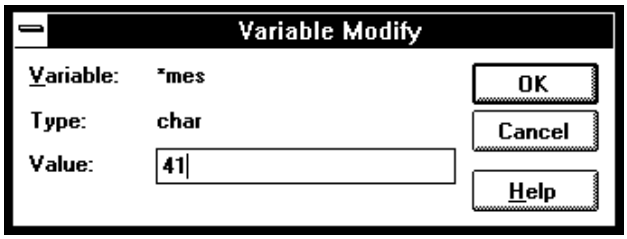
See Also

"To display a variable" and "To monitor a variable in the WatchPoint window" in the "Displaying and Editing Variables" section of the "Debugging Programs" chapter.

"Symbols" in the "Expressions in Commands" chapter.

Variable Modify Dialog Box

Choosing the Modify button in the Variable Edit dialog box opens the following dialog box, where you enter the new value and choose the OK button to confirm the new value.



Variable	Shows the variable to be edited.
Type	Indicates the type of the variable displayed in the Variable field.
Value	Lets you enter the new value of the variable.
OK	Replaces the contents of the specified variable with the specified value and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

See Also

"To edit a variable" in the "Displaying and Editing Variables" section of the "Debugging Programs" chapter.

Trace→Function Caller... (ALT, T, C)

Traces the caller of the specified function.

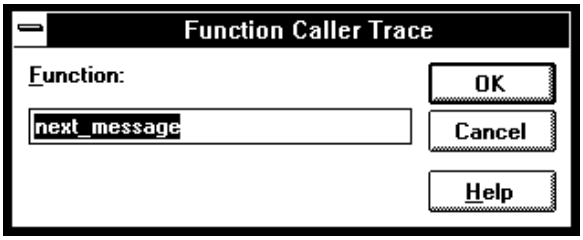
The function name can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

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	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.

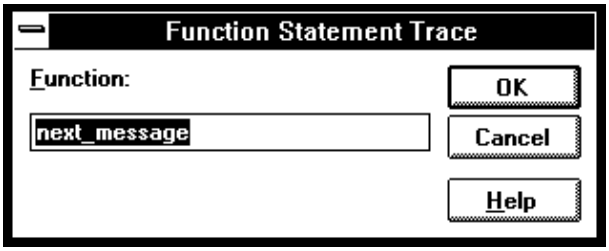
The function name can be selected from the another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

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:



Function	Lets you enter the function whose execution you want to trace.
OK	Traces within the specified function and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) FUNC(TION) STA(TEMENT) address
```

See Also

"To trace execution within a specified function" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Variable Access... (ALT, T, V)

Traces accesses to the specified variable.

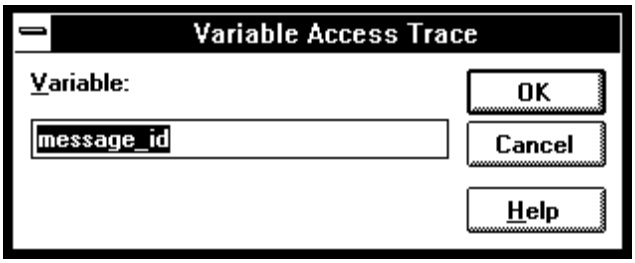
The variable name can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

You can specify any of the external or static variables, or the variables having a fixed address throughout the course of program execution.

The analyzer stores only accesses within the range of the variable and prestores execution states that occur before the access. These prestored states correspond to the statements that access the variable.

Variable Access Dialog Box

Choosing the Trace→Variable Access... (ALT, T, V) command opens the following dialog box:



Variable	Lets you enter the variable name.
OK	Traces accesses to the specified variable and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) VAR(IABLE) ACC(ESS) address
```

See Also

"To trace accesses to a specified variable" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→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 set up 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)

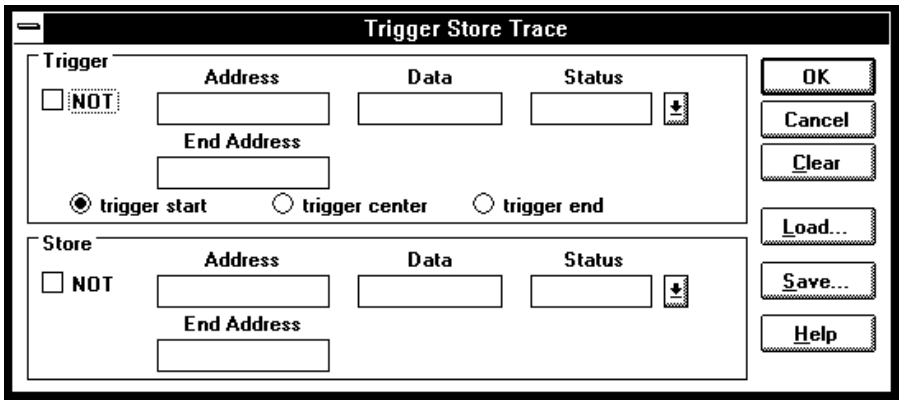
Trace→Trigger Store... (ALT, T, T)

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. See Understanding Addresss, Data, and Status for information and hints on setting up the A:D:S fields.

Trigger Store Trace Dialog Box

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



Trigger	This box groups the items that make up the trigger condition.
NOT	Specifies any state that does not match the Address, Data, and Status values.
Address	Specifies the address portion of the state qualifier.
End Address	Specifies the end address of an address range.
Data	Specifies the data portion of the state qualifier.
Status	Specifies the status portion of the state qualifier.
trigger start	Specifies that states captured after the trigger condition be stored in the trace buffer.
trigger center	Specifies that states captured before and after the trigger condition be stored in the trace buffer.
trigger end	Specifies that states captured before the trigger condition be stored in the trace buffer.

Store	This box groups the items that make up the store condition.
OK	Starts the specified trace and closes the dialog box.
Cancel	Cancels the trace setting and closes the dialog box.
Clear	Restores the dialog box to its default state.
Load...	Opens a file selection dialog box from which you select the name of a trace specification file previously saved from the Trigger Store Trace dialog box. Trace specification files have the extension ".TRC".
Save...	Opens a file selection dialog box from which you select the name of the trace specification file.

Command File Command

```
TRA(CE) LOA(D) filename
Loads the specified trace setting file.

TRA(CE) CUS(TOMIZE)
Traces program execution using the loaded trace setting
file.
```

See Also

"To set up a "Trigger Store" trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

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

Traces program execution as specified in the Find Then Trigger Trace dialog box.

This command lets you set up a two level sequential trace specification that works like this:

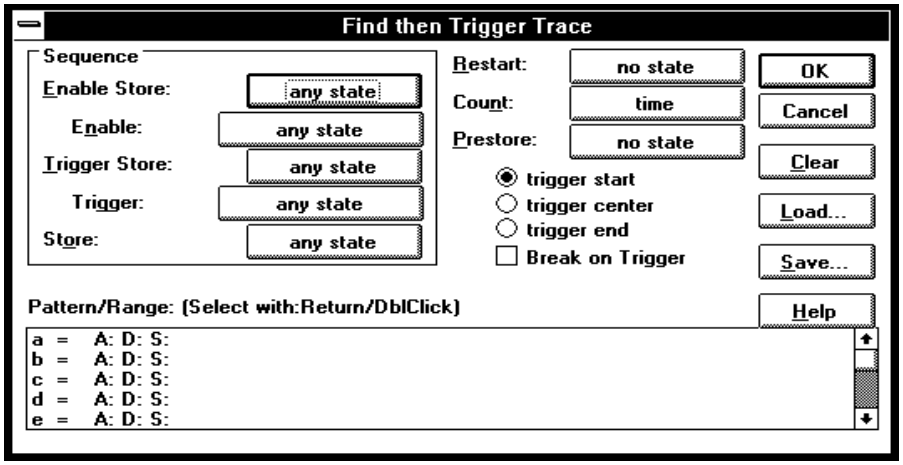
1. Once the trace starts, the analyzer stores (in the trace buffer) the states that satisfy the Enable Store condition while searching for a state that satisfies the Enable condition.
2. After the Enable condition has been found, the analyzer stores the states that satisfy the Trigger Store condition while searching for a state that satisfies the Trigger condition.
3. After the Trigger condition has been found, the analyzer stores the states that satisfy the Store condition.

If any state during the sequence satisfies the Restart condition, the sequence starts over.

You can enter address, data, and status values that qualify state(s) by setting up pattern or range resources. These patterns and range resources are used when defining the various conditions.

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.

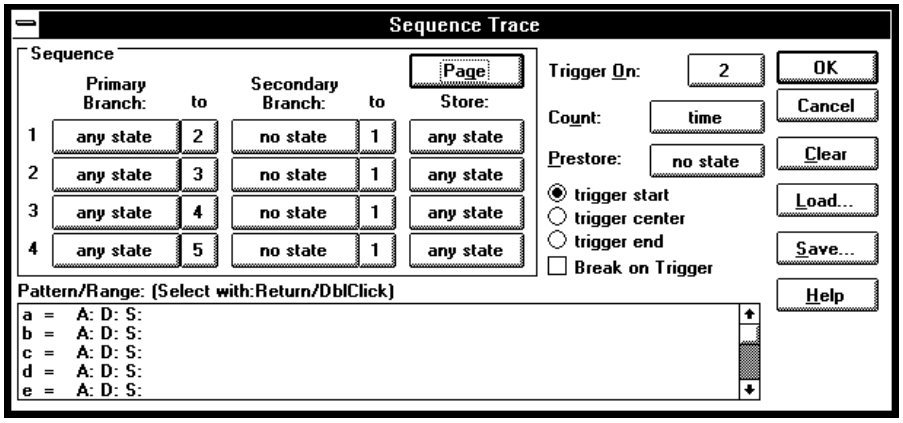
2 The analyzer stays at the next sequence level until its primary or secondary branch condition is met; then, the analyzer transfers to the sequence level specified by the "to" button.

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:



The Sequence group box specifies two types of branch conditions for transferring from one sequence level to another. It also specifies store conditions for each of sequence levels 1 through 8.

Primary Branch	Specifies the condition for transferring to the sequence level specified in the "to" text box.
Secondary Branch	Specifies the condition for transferring to the sequence level specified in the "to" text box. Secondary branches are used to do things like restart the sequence if a particular state is found.
Store	Specifies the states stored in the trace buffer at each sequence level.
Page	Toggles the display between sequence levels 1 through 4 and levels 5 through 8.
Trigger On	Specifies the sequence level whose entry triggers the analyzer. See the Sequence Number Dialog Box.

Count	Specifies whether time or the occurrences of a particular state are counted; you can also turn counts OFF. See the Condition Dialog Boxes.
Prestore	Qualifies the states that may be stored before each normally stored state. Up to two states may be prestored for each normally stored state. Prestored states can be used to show from where a function is called or a variable is accessed.
trigger start	The state that satisfies trigger condition is positioned at the start of the trace, and states that satisfy the store conditions will be stored after the trigger.
trigger center	The state that satisfies the trigger condition is positioned in the center of the trace, and states that satisfy the store conditions will be stored before and after the trigger.
trigger end	The state that satisfies the trigger condition is positioned at the end of the trace, and states that satisfy the store conditions will be stored before the trigger.
Break on Trigger	When selected, this option specifies that execution break into the monitor when the analyzer is triggered.
Pattern/Range	Specifies the trace patterns for the state conditions. Double-clicking the desired pattern in the Pattern/Range list box opens the Trace Pattern Dialog Box or the Trace Range Dialog Box, where you specify the desired trace pattern or range. Clicking the Primary Branch, Secondary Branch, Store, Count, or Prestore buttons causes the Condition Dialog Boxes to be opened. This dialog box lets you select or combine patterns or ranges to specify the condition.
OK	Starts the specified trace and closes the dialog box.
Cancel	Cancels trace setting and closes the dialog box.

Clear	Restores the dialog box to its default state.
Load...	Opens a file selection dialog box from which you select the name of a trace specification file previously saved from any of the trace setting dialog boxes. Trace specification files have the extension ".TRC".
Save...	Opens a file selection dialog box from which you select the name of the trace specification file.

Command File Command

```
TRA(CE) LOA(D) filename
Loads the specified trace setting file.

TRA(CE) CUS(TOMIZE)
Traces program execution using the loaded trace setting
file.
```

See Also

"To set up a "Sequence" trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Until Halt (ALT, T, U)

Traces program execution until the Trace→Halt (ALT, T, H) command is chosen.

This command is useful in tracing execution that leads to a processor halt or a break to the background monitor. Before executing the program, choose the Trace→Until Halt (ALT, T, U) command. Then, run the program. After the processor has halted or broken into the background monitor, choose the Trace→Halt (ALT, T, H) command to stop the trace. The execution that led up to the break or halt will be displayed.

Command File Command

```
TRA(CE) ALW(AYS)
```

See Also

"To trace until the command is halted" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Halt (ALT, T, H)

Stops a running trace.

This command stops a currently running trace whether the trace was started with the Trace→Until Halt (ALT, T, U) command or another trace command.

As soon as the analyzer stops the trace, stored states are displayed in the Trace window.

Command File Command

```
TRA(CE) STO(P)
```

See Also

"To stop a running trace" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Again (F7), (ALT, T, A)

Traces program execution using the last trace specification stored in the HP 64700.

If you haven't entered a trace command since you started the debugger, the last trace specification stored in the HP 64700 may be a trace specification set up by a different user; in this case, you cannot view or edit the trace specification.

Command File Command

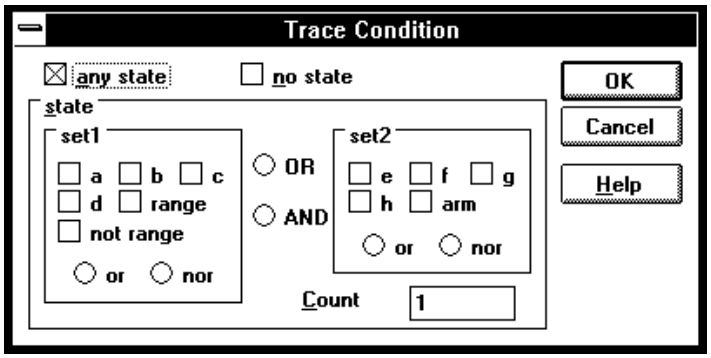
```
TRA(CE) AGA(IN)
```

See Also

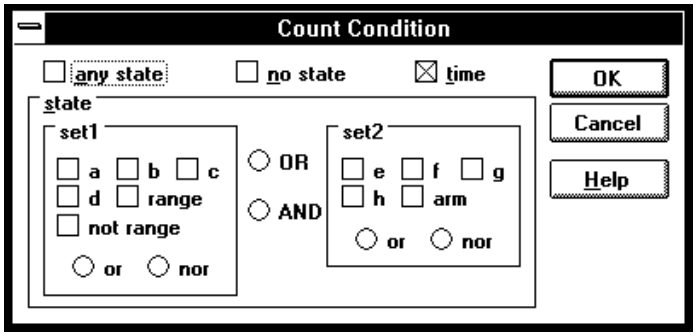
"To repeat the last trace" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Condition Dialog Boxes

Choosing the buttons associated with enable, trigger, primary branch, secondary branch, store, or prestore conditions opens the following dialog box:



Choosing the button associated with the count condition opens the following dialog box:



no state	No state meets the specified condition.
any state	Any state meets the specified condition.
time	The analyzer counts time for each state stored in the trace.

state — This group box lets you qualify the state that will meet the specified condition. You can qualify the state as one of the patterns "a" through "h," the "range," or the "arm," or you can qualify the state as a combination of the patterns, range, or arm by using the interset or intraset operators.

a b c d e f g h — The patterns that qualify states by identifying the address, data, and/or status values.

The values for a pattern are specified by selecting one of the patterns in the Pattern/Range list box and entering values in the Trace Pattern Dialog Box.

range — Identifies a range of address or data values.

The values for a range are specified by selecting the range in the Pattern/Range list box and entering values in the Trace Range Dialog Box.

not range — Identifies all values not in the specified range.

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

or/nor — You can combine patterns within the set1 or set2 group boxes with these logical operators.

You can create the AND and NAND operators by selecting NOT when defining patterns and applying DeMorgan's law (the / character is used to represent a logical NOT):

```
AND  A and B  =  /(/A or /B)  NOR
NAND  /(A and B) =  /A or /B  OR
```

OR/AND — You can combine patterns from the set1 and set2 group boxes with these logical operators.

Count	Appearing in Trace Condition dialog boxes, this value specifies the number of occurrences of the state that will satisfy the condition.
OK	Applies the state qualifier to the specified condition and closes the dialog box.
Cancel	Closes the dialog box.

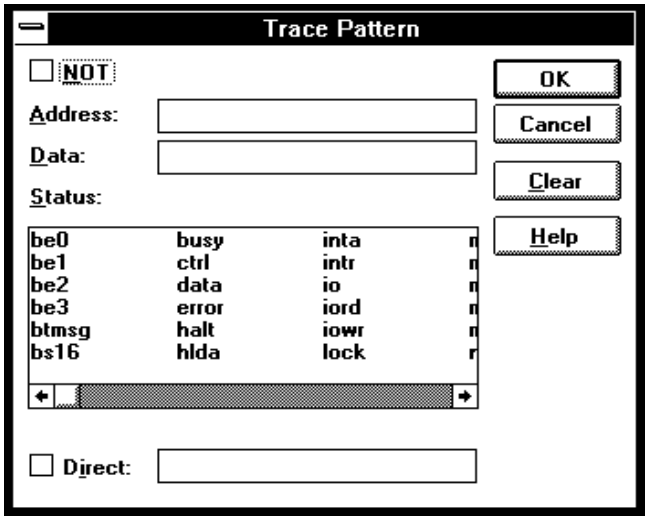
See Also

"To set up a "Find Then Trigger" trace specification", and "To set up a "Sequence" trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)
Trace→Sequence... (ALT, T, Q)

Trace Pattern Dialog Box

Selecting one of the patterns in the Pattern/Range list box opens the following dialog box:



NOT	Lets you specify all values other than the address, data, and/or status values specified.
Address	Lets you enter the address value for the pattern.
Data	Lets you enter the data value for the pattern.
Status	Lets you select the *status value* for the pattern.
Direct	Lets you enter a status value other than one of the predefined status values.
Clear	Clears the values specified for the pattern.
OK	Applies the values specified for the pattern, and closes the dialog box.

Cancel	Closes the dialog box.

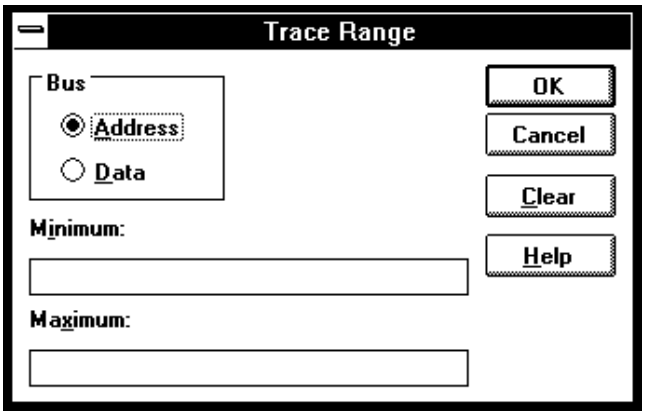
See Also

"To set up a "Find Then Trigger" trace specification", and "To set up a "Sequence" trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)
Trace→Sequence... (ALT, T, Q)

Trace Range Dialog Box

Selecting the range in the Pattern/Range list box opens the following dialog box:



Address	Selects a range of address values.
Data	Selects a range of data values.
Minimum	Lets you enter the minimum value for the range.
Maximum	Lets you enter the maximum value for the range.
OK	Applies the values specified for the range, and closes the dialog box.
Cancel	Closes the dialog box.
Clear	Clears the values specified for the range.

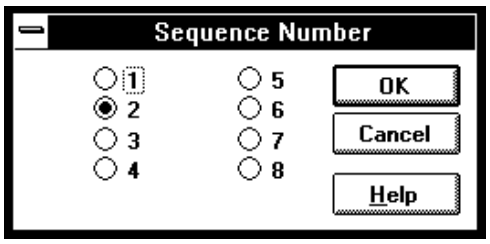
See Also

"To set up a "Find Then Trigger" trace specification", and "To set up a "Sequence" trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Find Then Trigger... (ALT, T, D)
Trace→Sequence... (ALT, T, Q)

Sequence Number Dialog Box

Choosing the buttons associated with "to" or Trigger On opens the following dialog box:



1-8	These options specify the sequence level.
OK	Applies the selected sequence level and closes the dialog box.
Cancel	Closes the dialog box.

See Also

"To set up a "Sequence" trace specification" in the "Setting Up Custom Trace Specifications" section of the "Debugging Programs" chapter.

Trace→Sequence... (ALT, T, Q)

RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D)

Activates the real-time mode.

When the user program is running in real-time mode, no command that would normally cause temporary suspension of program execution is allowed. Also, the system hides:

- The Register window.
- Target system memory in the Memory window.
- Target system I/O locations in the I/O window.
- Target system memory variables in the WatchPoint window.
- Target system memory in the Source window.

While the processor is in the RUNNING REALTIME IN USER PROGRAM state, no display or modification is allowed for the contents of target system memory or registers. Therefore, before you can display or modify target system memory or processor registers, you must use the Execution→Break (ALT, E, B) command to stop user program execution and break into the monitor.

Command File Command

```
MOD(E) REA(LTIME) ON
```

See Also

"To allow or deny monitor intrusion" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Monitor Intrusion→Allowed (ALT, R, T, A)

Deactivates the real-time mode.

Commands that cause temporary breaks to the monitor during program execution are allowed.

Command File Command

```
MOD(E) REA(LTIME) OFF
```

See Also

"To allow or deny monitor intrusion" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→I/O Polling→ON (ALT, R, I, O)

Enables access to I/O.

Command File Command

```
MOD(E) IOG(UARD) OFF
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→I/O Polling→OFF (ALT, R, I, F)

Disables access to I/O.

When polling is turned OFF, values in the I/O window are updated on entry to the monitor. When monitor intrusion is not allowed during program execution, the I/O window is not updated and contents are replaced by dashes (-).

Command File Command

```
MOD(E) IOG(UARD) ON
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Watchpoint Polling→ON (ALT, R, W, O)

Turns ON polling to update values displayed in the WatchPoint window.

When polling is turned ON, temporary breaks in program execution occur when the WatchPoint window is updated.

Command File Command

```
MOD(E) WAT(CHPOLL) ON
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Watchpoint Polling→OFF (ALT, R, W, F)

Turns OFF polling to update values displayed in the WatchPoint window.

When polling is turned OFF, values in the WatchPoint window are updated on entry to the monitor. When monitor intrusion is not allowed during program execution, the WatchPoint window is not updated and contents are replaced by dashes (-).

Command File Command

```
MOD(E) WAT(CHPOLL) OFF
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Memory Polling→ON (ALT, R, M, O)

Turns ON polling to update target memory values displayed in the Memory window.

When polling is turned ON, temporary breaks in program execution occur when target system memory locations in the Memory window are updated. When monitor intrusion is not allowed during program execution, the contents of target memory locations are replaced by dashes (-).

Also, when polling is turned ON, you can modify the addresses displayed or contents of memory locations by double-clicking on the address or value, using the keyboard to type in the new address or value, and pressing the Enter key.

Command File Command

```
MOD(E) MEM(ORYPOLL) ON
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

RealTime→Memory Polling→OFF (ALT, R, M, F)

Turns OFF polling to update target memory values displayed in the Memory window.

When polling is turned OFF, values in the Memory window are updated on entry to the monitor.

Also, when polling is turned OFF, you cannot modify the addresses displayed or contents of memory locations by double-clicking on the address or value.

Command File Command

```
MOD(E) MEM(ORYPOLL) OFF
```

See Also

"To turn polling ON or OFF" in the "Setting the Real-Time Options" section of the "Configuring the Emulator" chapter.

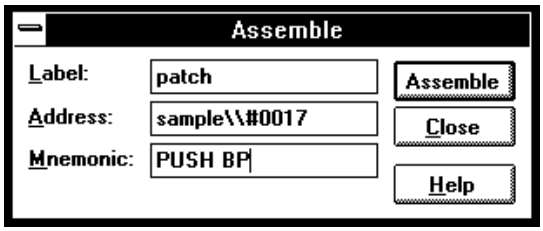
Assemble... (ALT, A)

In-line assembler.

This command lets you modify programs by specifying assembly language instructions which are assembled and loaded into program memory.

Assembler Dialog Box

Choosing the Assemble... (ALT, A) command opens the following dialog box:



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

Command File Command

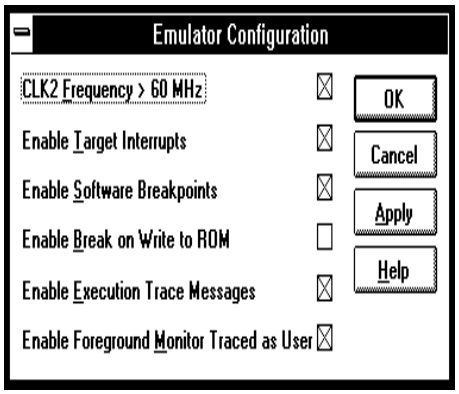
```
ASM address label "inst_string"
```

Settings→Emulator Config→Hardware... (ALT, S, E, H)

Specifies the emulator configuration.

Hardware Config Dialog Box

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



CLK2 Frequency > 60 MHz	Specifies whether one wait state should be added for accesses to memory mapped into 4-Mbyte SIMMs. Note that CLK2 is the oscillator frequency to the 80386. It is twice the frequency of the usually-quoted value. For example, a "33 MHz 80386" has a CLK2 of 66 MHz.
Enable Target Interrupts	Enables or disables target interrupts. If interrupts are disabled, no interrupts (INT or NMI) are passed to the processor. If enabled, interrupts are passed when executing user code or when using the foreground monitor. In any case, when using the background monitor, interrupts will be ignored while in the monitor.
Enable Software Breakpoints	Enables or disables software breakpoints. If disabled, you cannot set any breakpoints. If enabled, you can set

	software breakpoints. If software breakpoints are set, the emulator will take a longer time to leave the RESET state because it must break into the monitor to enable the software breakpoints each time it leaves the RESET state.
Enable Break on Write to ROM	Enables or disables breaks to the monitor when the user program writes to memory mapped as ROM.
Enable Execution Trace Messages	Enables or disables branch trace messages and task switch messages: If enabled, every time the processor does a branch, it will emit the target address of the branch. See Understanding 80386 Analysis for more information about how to use branch trace messages. Also, any task switch will emit a task switch message telling you what the old task was and what the new task is.
Enable Foreground Monitor Traced as User	Enables or disables tracing when execution is in the foreground monitor. When using a foreground monitor with this selected, all foreground monitor cycles will be captured in the trace memory by the emulation-bus analyzer. This is useful when you are having problems with an interrupt routine and you want to trace that routine even if it occurs during execution in the foreground monitor. If this is not selected, and you have chosen Settings→Extended→Trace Cycles→User, the analyzer will capture nothing between the time the foreground monitor is entered and the time you begin a run of your user program again. This prevents capture of interrupt routines executed while in the foreground monitor. When using the background monitor, this has no effect.
OK	Stores the current modification and closes the dialog box.
Cancel	Cancels the current modification and closes the dialog box.
Apply	Loads the configuration settings into the emulator.

Command File Command

```
CON(FIG) FASTCLK ENABLE

CON(FIG) FASTCLK DISABLE

CON(FIG) INT(RS) ENABLE

CON(FIG) INT(RS) DISABLE

CON(FIG) WRROM ENABLE

CON(FIG) WRROM DISABLE

CON(FIG) BKPTS ENABLE

CON(FIG) BKPTS DISABLE

CON(FIG) EMSGS ENABLE

CON(FIG) EMSGS DISABLE
```

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

```
CON(FIG) STA(RT)
Starts the configuration option command section.

CON(FIG) END
Ends the configuration option command section.
```

See Also

"Setting the Hardware Options" in the "Configuring the Emulator" chapter.

"Tracing Program Execution" in the "Debugging Programs" chapter for useful combinations of the Settings→Extended→Trace Cycles command and the Enable Foreground Monitor Traced as User selection.

Settings→Emulator Config→Memory Map... (ALT, S, E, M)

Maps memory ranges.

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

You can map ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

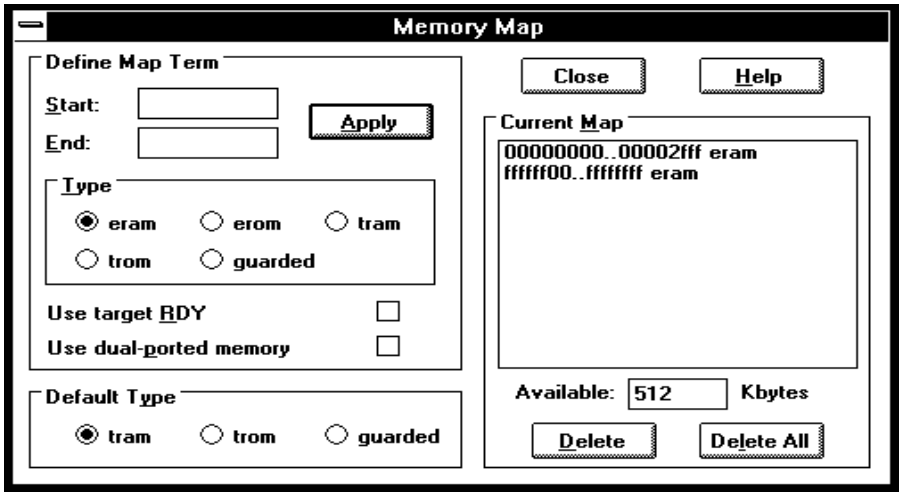
Guarded memory accesses cause emulator execution to break into the monitor program.

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 will modify memory. Writes by user code to target system memory locations that are mapped as ROM or guarded memory may result in a break to the monitor but they are not inhibited (that is, the write still occurs).

Memory Map Dialog Box

Choosing the Settings→Emulator Config→Memory Map... (ALT, S, E, M) command opens the following dialog box:



Start	Specifies the starting address of the address range to be mapped.
End	Specifies the end address of the address range to be mapped.
Type	Lets you select the memory type of the specified address range.
Use Target RDY	Specifies that emulation memory accesses in the range be synchronized to the target system RDY signal.

Use dual-ported memory	Specifies that this memory range will be placed in the 8K of dual-ported memory. Note that you can only map one address range to this memory. RTC can access this memory without breaking into the monitor when the processor is running and not in the HALT or SHUTDOWN state. If the processor is in the HALT or SHUTDOWN state, however, dual-port memory cannot be accessed. In that case, the emulator will break into the monitor to read the memory. To prevent the break into the monitor in this case, choose Realtime→Monitor Intrusion→Disallowed.
Apply	Maps the address range specified in the Define Map Term group box.
Default Type	Specifies the type of unmapped 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 attributes
Maps the specified address range with the specified
memory type.  When mapping emulation memory ranges, the
attributes can be a comma-separated list including:
```

dp	map the address range to dual-port memory.
trdy	memory accesses in the range will be synchronized to the target system READY# signal.

```
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

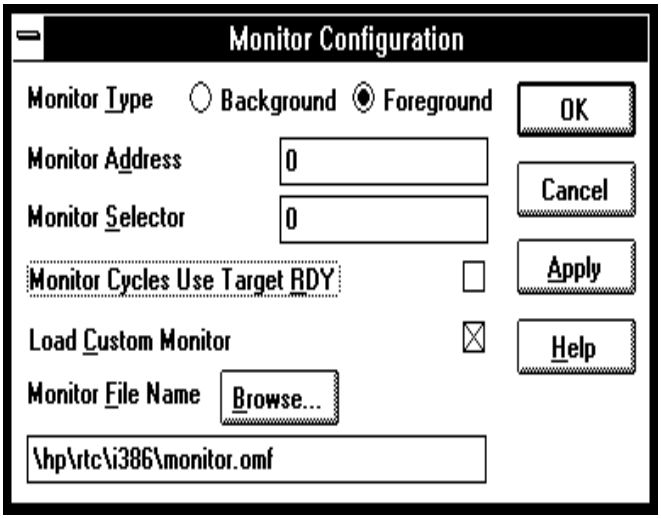
"Mapping Memory" in the "Configuring the Emulator" chapter.

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:



Monitor Type	Lets you choose between a background monitor and a foreground monitor.
Monitor Address	Specifies the starting address of the foreground monitor program. The address must reside on a 16-Kbyte boundary (in other words, a multiple of 4000H) and must be specified in hexadecimal. In order for the foreground monitor to run in real mode, the base address must be limited to 000fc000 hex. Higher addresses can be selected if the target program always runs in protected mode. However, any attempt to break before protected mode is enabled will result in the background monitor being used (target interrupts will be blocked).

Monitor Selector	Selects the GDT descriptor for the foreground monitor code segment. The foreground monitor is interruptable and is designed to run in both real and protected modes based on the current state of the processor. In order to run in protected mode, a GDT entry must be reserved to define the code segment for the monitor. The specified value must be a multiple of 8, greater than 0 and less than the limit defined in GDTR.
Monitor Cycles Use Target RDY	Specifies whether monitor cycles should be synchronized to the target system (in other words, whether the emulation and target system READY# should be interlocked on accesses to the monitor memory block).
Load Custom Monitor	Specifies whether the default foreground monitor (resident in the emulator firmware) or a custom monitor should be used.
Monitor File Name	When using a customized foreground monitor program, this text box lets you enter the name of the object file. An example foreground monitor is provided with the debugger in the C:\HP\RTC\I386\MONITOR directory (if C:\HP\RTC\I386 was the installation path chosen when installing the debugger software). The file is named I386DX.ASM. The foreground monitor is automatically loaded after each Execution→Reset (ALT, E, E) command.
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.
Cancel	Cancels the monitor configuration and closes the dialog box.
Apply	Loads the configuration settings into the emulator.

Command File Command

```
MON(ITOR) TYPE FOREGROUND
Selects the foreground monitor.

MON(ITOR) TYPE BACKGROUND
Selects the background monitor.

MON(ITOR) ADD(RESS) address
Specifies the monitor's base address.

MON(ITOR) SELECTOR selector
specifies the monitor's selector.

MON(ITOR) TRDY ENA(BLE)
Enables synchronization of monitor cycles to the target
system (that is, interlock the emulation and target
system RDY on accesses to the monitor memory block).

MON(ITOR) TRDY DIS(ABLE)
Disables synchronization of monitor cycles to the
target system.

MON(ITOR) FIL(ENAME) NONE
Specifies using the built-in foreground monitor.

MON(ITOR) FIL(ENAME) 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:

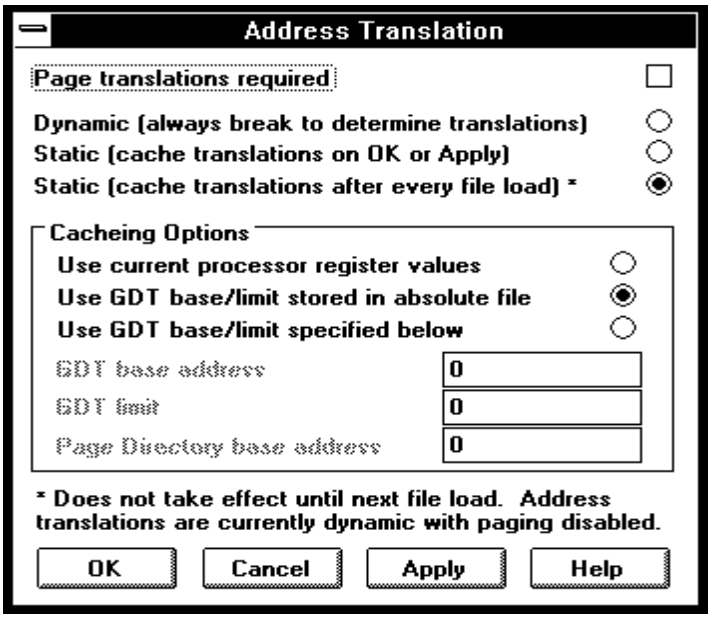
```
MON(ITOR) STA(RT)
Starts the monitor option command section.

MON(ITOR) END
Ends the monitor option command section.
```

See Also

"Selecting the Type of Monitor" in the "Configuring the Emulator" chapter.

Settings→Emulator Config→Address Translation... (ALT, S, E, A)

Choosing the Settings→Emulator Config→Address Translations... (ALT, S, E, A) command opens the following dialog box:



Page translations required — Specifies that paging is used by your target system; therefore, any virtual-to-physical translation will need to traverse the page tables.

HP strongly recommends you not use this unless your target system uses paging because your system performance will be improved if the page tables do not need to be traversed every time a translation occurs.

Dynamic (always break to determine translations)	Specifies that the emulator will temporarily break from execution of your target program into the monitor to do a translation. With this selection, the translation will always be accurate for the current state of the processor and the current GDT (if in protected mode). Choose "Dynamic" if your GDT tables change frequently. The only negative aspect of making this choice is that you cannot set up the analyzer or modify and display memory using protected-mode addresses when the processor is RESET or in real mode. You must use physical addresses in these cases.
Static (cache translations on OK or Apply)	Specifies that cached translations will be used, and that the source for the cache will be read from the 80386 when the OK or Apply button is pressed. The "Cacheing Options", below, will be consulted to determine the location of the GDT and page tables.
Static (cache translations after every file load)	Specifies that cached translations will be used, and that the source for the cache is from an object file. When a file is loaded, the cache will be updated. The "Cacheing Options", below, will be consulted to determine the location of the GDT and page tables within the absolute file. Note that when this is chosen, the current translation scheme is used until the next File→Load Object... command is given. For example, if the mode is "dynamic" when this is chosen, address translations will continue to be dynamic until the next successful File→Load Object... command.

Cacheing Options

Use current processor register values	Specifies that the current register values for GDTR, CR0, and CR3 are read, then their values are used to cache GDT and LDT tables as well as page tables.

Use GDT base/limit stored in absolute file	Specifies that the GDT location is stored in the absolute file and is marked as such. Some builders provide this data and some do not. If you try this and fail, use the next option.
Use GDT base/limit specified below	Specifies that the GDT address and size will be taken from the values in the edit boxes below:
	GDT base address Specifies the base address of the GDT. Note that this address must be a linear address (not virtual).
	GDT limit Specifies the limit of the GDT; it must be a multiple of 8 minus 1 (bytes). For example, if there were four entries in the GDT, the value would be 31 (01F); (8*4)-1.
	Page Directory base address Specifies the base address of the page table. Note that this address must be a physical address (not virtual or linear), and must be a multiple of 4K (it must end in 000, when entered in hex).

Command File Command

```
ADDRTRAN PAGING ON
Specify that paging is enabled, so page tables must be
traversed in order to translate linear (and virtual)
addresses to physical.

ADDRTRAN PAGING OFF
Specify that paging is disabled.

ADDRTRAN METHOD DYNAMIC
Specify that dynamic address translations should be
used.

ADDRTRAN METHOD STATICOKAY
Specify that static address translations should be
used, and cache the GDT & page tables immediately
```

```
ADDRTRAN METHOD STATICFILE
Specify that static address translations should be
used, and cache the GDT & page tables whenever a file
is loaded into the emulator.

ADDTRAN CACHE CURRENT
Specify that the current register values for GDTR, CR0
and CR3 are read, and then their values are used to
cache GDT and LDT tables as well as page tables.

ADDRTRAN CACHE FROMOMF
Specify that when cacheing the GDT, the base and limit
of the GDT is to be taken from the OMF386 file loaded
into the emulator.

ADDTRAN CACHE FROMVAL
Specify that the GDT address and size will be taken
from the values specified in "ADDRTRAN GDTBASE base"
and "ADDRTRAN GDTLIMIT limit", and in "ADDRTRAN PDBASE
base" (if applicable).

ADDRTRAN GDTBASE base
Specify that when cacheing the GDT, the address of the
GDT is "base".

ADDRTRAN GDTLIMIT limit
Specify that when cacheing the GDT, the limit of the
GDT is "limit".

ADDRTRAN PDBASE base
Specify that when cacheing the page tables, the address
of the page table is "base".
```

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

```
ADDRTRAN STA(RT)
Starts the address translation command section.

ADDRTRAN END
Ends the address translation command section.
```

See Also

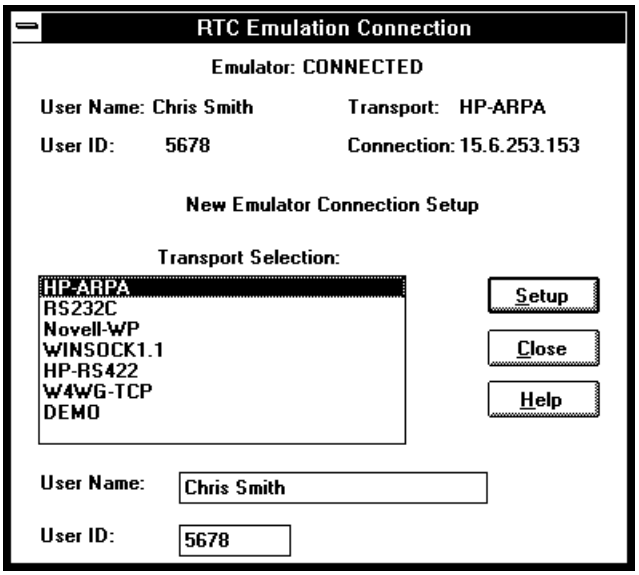
"Selecting how Address Translations work" in the "Configuring the Emulator" chapter.

Settings→Communication... (ALT, S, C)

Choosing this command opens the RTC Emulation Connection Dialog Box which lets you identify and set up the communication channel between the personal computer and the HP 64700.

RTC Emulation Connection Dialog Box

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



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 LAN Setup dialog boxes, enter the IP address or network name of the HP 64700.

In the RS232C Setup dialog box, select the baud rate and the name of the port (for example, COM1, COM2, etc.) to which the HP 64700 is connected.

In the HP-RS422 Setup dialog box, select the baud rate and specify the I/O address you want to use for the HP 64037 card. The I/O address must be a hexadecimal number from 100H through 3F8H, ending in 0 or 8, that does not conflict with other cards in your PC.

The Connect button in any of these Setup dialog boxes starts the debugger with the specified communication settings.

Close — Either closes the Real-Time C Debugger, if the current connection failed, or simply closes the dialog box.

The Real-Time C Debugger does not allow you to change connection or transport information without leaving the debugger and 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.

Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O)

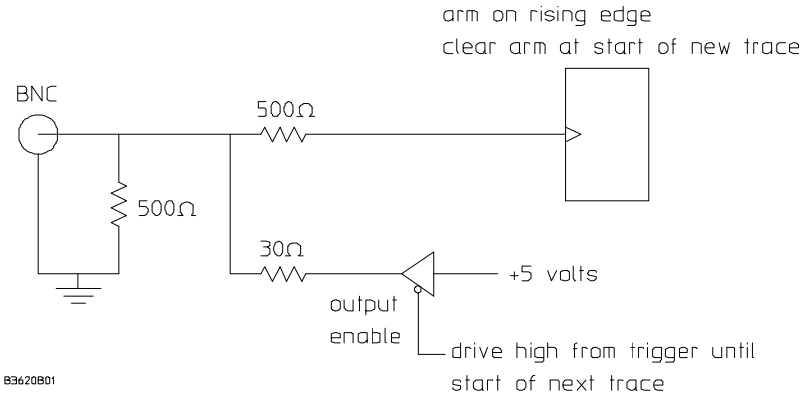
Specifies that the analyzer trigger signal be driven on the BNC port.

Selecting the emulator BNC port for output enables the trigger signals to be fed to external devices (for example, logic analyzers) during tracing.

CAUTION

Do not drive the BNC beyond the range of 0 to 5 volts. Doing so may cause permanent damage to the HP 64700.

The BNC's drivers can drive 50 ohm loads. The following is a logical diagram of the BNC connection. The physical implementation and values of resistors are not exact. This diagram is just to help you understand the BNC interface:



When a trace starts, it stops driving the output (so if nothing else is driving the line, it will fall low due to the 500 ohm pull-down resistor).

When the trigger point is found, the BNC starts driving the output high. It will stay high until the start of the next trace.

Command File Command

```
MOD(E) BNC OUT(PUT_TRIGGER)
```

See Also

"To output the trigger signal on the BNC port" in the "Setting Up the BNC Port" section of the "Configuring the Emulator" chapter.

Settings→BNC→Input to Analyzer Arm (ALT, S, B, I)

Allows the analyzer to receive an arm signal from the BNC port.

This command allows an external trigger signal to be used as an arm (enable) condition for the internal analyzer. The internal analyzer will arm (or enable) on a positive edge TTL signal.

CAUTION Do not drive the BNC beyond the range of 0 to 5 volts. Doing so may cause permanent damage to the HP 64700.

You can use the arm condition when setting up custom trace specifications with the Trace→Find Then Trigger... (ALT, T, D) or Trace→Sequence... (ALT, T, Q) commands. For example, you can trigger on the arm condition or enable the storage of states on the arm condition. The "arm" condition may be selected in "set2" of the Trace Condition or Count Condition dialog boxes.

The BNC port is internally terminated with about 500 ohms; if using a 50 ohm driver, use an external 50 ohm termination (such as the HP 10100C 50 Ohm Feedthrough Termination) to reduce bouncing and possible incorrect triggering.

Command File Command

```
MOD(E) BNC INP(UT_ARM)
```

See Also

Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O) for a logical schematic of the BNC interface.

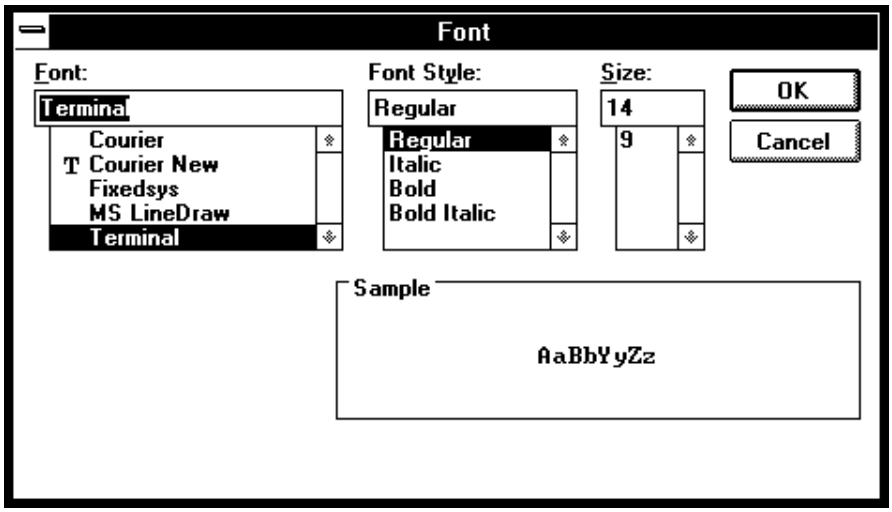
"To receive an arm condition input on the BNC port" in the "Setting Up the BNC Port" section of the "Configuring the Emulator" chapter.

Settings→Font... (ALT, S, F)

Selects the fonts used in the debugger windows.

Font Dialog Box

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



Font	Lets you select the font to be used in the Real-Time C Debugger interface. The "T" shaped icon indicates a TrueType font.
Font Style	Lets you select the typeface, for example, regular, bold, italic, etc.
Size	Lets you select the size of the characters.
Sample	Shows you what the selected font looks like.
OK	Sets the font, and closes the dialog box.
Cancel	Cancels font setting, and closes the dialog box.

See Also

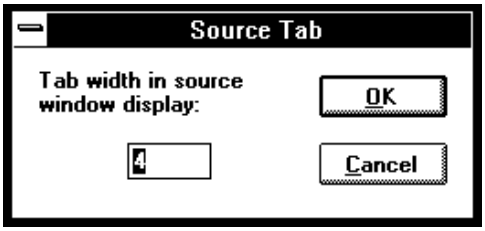
"To change the debugger window fonts" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Settings→Tabstops... (ALT, S, T)

Sets the number of spaces between tab stops.

Source Tab Dialog Box

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



Tab width in source window display	Enter the number of spaces between tab stops. This also affects the tab width for source lines in the Trace window.
OK	Sets the tab stops, and closes the dialog box.
Cancel	Cancels tab stop setting, and closes the dialog box.

See Also

"To set tab stops in the Source window" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O)

Symbol database search is case sensitive.

Command File Command

```
MOD(E) SYM(BOLCASE) ON
```

See Also

Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F)

Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F)

Symbol database search is not case sensitive.

If there are case conflicts (for example, FOO and foo), no warning is given, and you cannot predict which symbol will be used. The symbol that is used depends on what type of symbols FOO and foo are and how they were input by the symbol section of the object file.

Command File Command

```
MOD(E) SYM(BOLCASE) OFF
```

See Also

Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O)

Settings→Extended Settings→Trace Cycles→User (ALT, S, X, T, U)

Traces foreground emulation microprocessor operation.

This is the normal setting.

Command File Command

```
MOD(E) TRA(CECLOCK) USE(R)
```

See Also

Settings→Extended Settings→Trace Cycles→Monitor (ALT, S, X, T, M)
Settings→Extended Settings→Trace Cycles→Both (ALT, S, X, T, B)

Settings→Extended Settings→Trace Cycles→Monitor (ALT, S, X, T, M)

Traces background emulation microprocessor operation.

This is rarely a useful setting when debugging programs.

Command File Command

```
MOD(E) TRA(CECLOCK) BAC(KGROUND)
```

See Also

Settings→Extended Settings→Trace Cycles→User (ALT, S, X, T, U)
Settings→Extended Settings→Trace Cycles→Both (ALT, S, X, T, B)

Settings→Extended Settings→Trace Cycles→Both (ALT, S, X, T, B)

Traces both foreground and background emulation microprocessor operation.

Command File Command

```
MOD(E) TRA(CECLOCK) BOT(H)
```

See Also

Settings→Extended Settings→Trace Cycles→User (ALT, S, X, T, U)
Settings→Extended Settings→Trace Cycles→Monitor (ALT, S, X, T, M)

Settings→Extended Settings→Load Error Abort→ON (ALT, S, X, L, O)

An error during an object file or memory load causes an abort.

Normally, when an error occurs during an object file or memory load, you want the load to stop so that you can fix whatever caused the error.

Command File Command

```
MOD(E) DOW(NLOAD) ERR(ABORT)
```

See Also

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

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

An error during an object file or memory load does not cause an abort.

If you expect certain errors during an object file or memory load, for example, if part of the file is located at "guarded" memory or "target ROM," you can choose this command to continue loading in spite of the errors.

Command File Command

```
MOD(E) DOW(NLOAD) NOE(RRABORT)
```

See Also

Settings→Extended Settings→Load Error Abort→ON (ALT, S, X, L, O)

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

You are prompted for source file paths.

When the debugger cannot find source file information for the Source or Trace windows, it may prompt you for source file paths depending on the MODE SOURCE setting.

Command File Command

```
MOD(E) SOU(RCE) ASK(PATH)
```

See Also

Settings→Extended Settings→Source Path Query→OFF (ALT, S, X, S, F)

Settings→Extended Settings→Source Path Query→OFF (ALT, S, X, S, F)

You are not prompted for source file paths.

You can turn off source path prompting, for example, to avoid annoying dialog interactions when tracing library functions for which no source files are available.

Command File Command

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

See Also

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

Window→Cascade (ALT, W, C)

Arranges, sizes, and overlaps windows.

Windows are sized, evenly, to be as large as possible.

Window→Tile (ALT, W, T)

Arranges and sizes windows so that none are overlapped.

Windows are sized evenly.

Window→Arrange Icons (ALT, W, A)

Rearranges icons in the Real-Time C Debugger window.

Icons are distributed evenly along the lower edge of the Real-Time C Debugger window.

Window→1-9 (ALT, W, 1-9)

Opens the window associated with the number.

The nine most recently opened windows appear in the menu list. If the window you wish to open is not on the list, choose the Window→More Windows... (ALT, W, M) command.

Windows are closed just as are ordinary MS Windows, that is, by opening the control menu and choosing Close or by pressing CTRL+F4.

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

Command File Command

```
DIS(PLAY) window-name
Opens the specified window.  Use the first three
characters of the window name, or, if the window name
is "Basic Registers", user "REG".

ICO(NIC) window-name
Closes the specified window.  Use the first three
characters of the window name, or, if the window name
is "Basic Registers", user "REG".
```

See Also

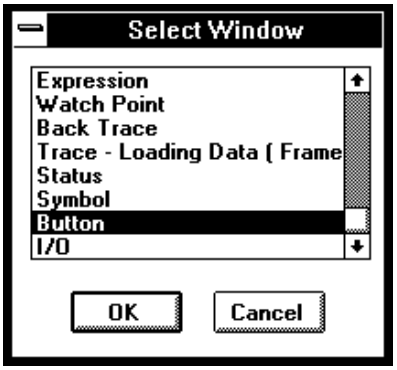
"To open debugger windows" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Window→More Windows... (ALT, W, M)

Presents a list box from which you can select the window to be opened.

Select Window Dialog Box

Choosing the Window→More Windows... (ALT, W, M) command opens the following dialog box:



OK	Opens the window selected in the list box.
Cancel	Closes the dialog box.

Command File Command

```
DIS(PLAY) window-name
Opens the specified window.

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.

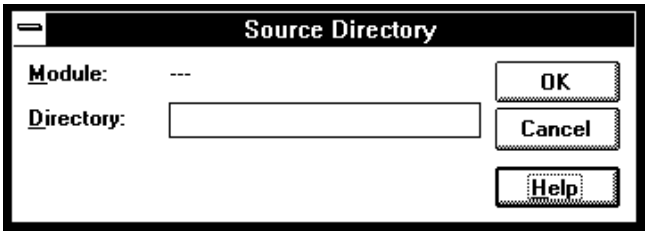
Help→About Debugger/Emulator... (ALT, H, D)

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.

You may not wish to have this dialog box open. There is a way to prevent it from opening. If you select Settings→Extended→Source Path Query→OFF, this dialog box will not open. If you wish to have this dialog box open when the source file associated with a symbol cannot be found, select Settings→Extended→Source Path Query→ON.

9

Window Control Menu Commands

Window Control Menu Commands

This chapter describes the commands that can be chosen from the *control menus* in debugger windows.

- Common Control Menu Commands
- Button Window Commands
- Expression Window Commands
- I/O Window Commands
- Memory Window Commands
- GDT/LDT/IDT Window Commands
- Register 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) BUT(TON)

COP(Y) EXP(RESSION)

COP(Y) IO

COP(Y) MEM(ORY)

COP(Y) REG(ISTER)

COP(Y) SOU(RCE)

COP(Y) WAT(CHPOINT)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Copy→Destination... (ALT, -, P, D)

Names the listing file to which debugger information may be copied.

This command opens a file selection dialog box from which you can select the listing file. Listing files have the extension ".LST".

Command File Command

```
COP(Y) TO filename
```

See Also

"To change the list file destination" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Button Window Commands

This section describes the following command:

- Edit... (ALT, -, E)

Edit... (ALT, -, E)

Lets you define and label buttons in the Button window.

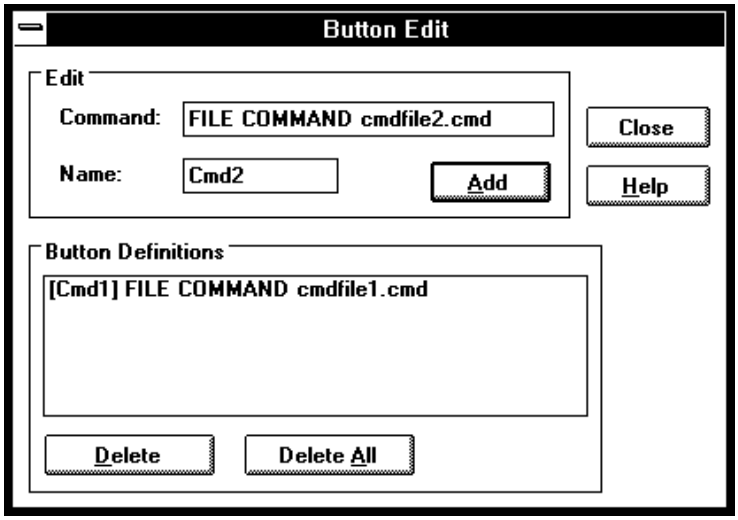
You can set up buttons to execute commonly used commands or command files.

Note that the Copy→Window command will generate a listing file that contains a header followed by commands needed to recreate the buttons. By removing the header, this file may be used as a command file.

Alternatively, you can log commands to a command file as you edit the buttons (refer to To create a command file in the "Using Command Files" section of the "Using the Debugger Interface" chapter). To recreate the buttons, just run the command file that you created while editing the buttons.

Button Edit Dialog Box

Choosing the Edit... (ALT, -, E) command opens the following dialog box:



Command	Specifies the command to be associated with the button. Command syntax is described at the bottom of most help topics under the "Command File Command" heading. Also, look in the Command File and Macro Command Summary chapter in the "Reference" part. You can only enter a single command here; if you want a series of commands to be executed when this button is used, put them in a command file and use the command "FILE COMMAND filename," where "filename" is the name of your command file.
Name	Specifies the button label to be associated with the command.
Add	Adds the button to the button window.
Button Definitions	Lists the currently defined buttons. You can select button definitions for deletion by clicking on them.

Delete	Deletes the button definition selected in the Button Definitions list box.
Delete All	Deletes all buttons from the Button window.
Close	Closes the dialog box.

Command File Command

```
BUTTON label "command"
```

See Also

"To create buttons that execute command files" in the "Using Command Files" section of the "Using the Debugger Interface" chapter.

Expression Window Commands

This section describes the following commands:

- Clear (ALT, -, R)
- Evaluate... (ALT, -, E)

Clear (ALT, -, R)

Erases the contents of the Expression window.

Command File Command

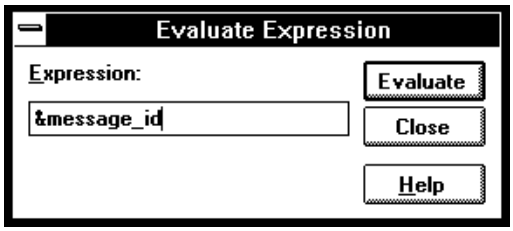
```
EVA(LUATE) CLE(AR)
```

Evaluate... (ALT, -, E)

Evaluates expressions and displays the results in the Expression window.

Evaluate Expression Dialog Box

Choosing the Evaluate... (ALT, -, E) command opens the following dialog box:



Expression	Lets you enter the expression to be evaluated.
Evaluate	Makes the evaluation and places the results in the Expression window.
Close	Closes the dialog box.

Command File Command

```
EVA(LUATE) address

EVA(LUATE) "strings"
```

See Also

"Symbols" in the "Expressions in Commands" chapter.

I/O Window Commands

This section describes the following command:

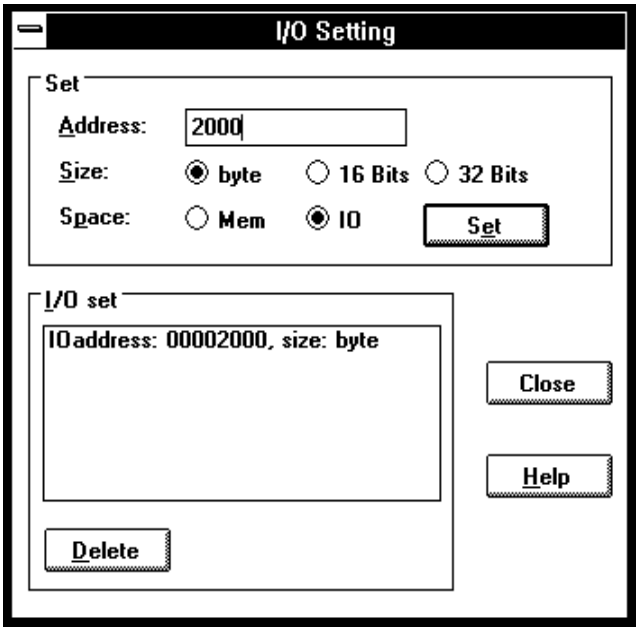
- Define... (ALT, -, D)

Define... (ALT, -, D)

Adds or deletes memory mapped I/O locations from the I/O window.

I/O Setting Dialog Box

Choosing the Edit→Definition... command opens the following dialog box:



Address — Specifies the address of the I/O location to be defined.

Size	Specifies the data format of the I/O location to be defined. You can select the Byte, 16 Bits, or 32 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 multicolumn format.

Command File Command

```
MEM(ORY) BLO(CK)
```

Display→Byte (ALT, -, D, Y)

Displays memory contents as bytes.

Command File Command

```
MEM(ORY) BYTE
```

Display→16 Bit (ALT, -, D, 1)

Displays memory contents as 16-bit values.

Command File Command

```
MEM(ORY) WORD
```

Display→32 Bit (ALT, -, D, 3)

Displays memory contents as 32-bit values.

Command File Command

```
MEM(ORY) LONG
```

Search... (ALT, -, R)

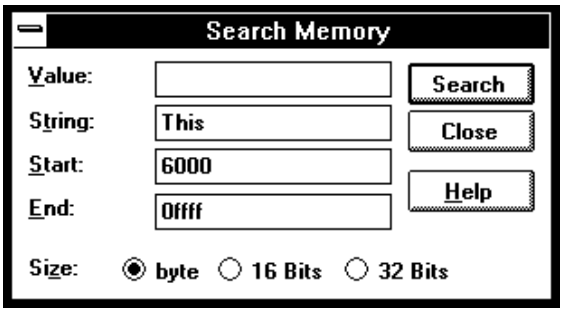
Searches for a value or string in a range of memory.

When the value or string is found, the location is displayed in the Memory window. Choose the Window→Memory command to open the window.

The value or string can be selected from another window (in other words, copied to the clipboard) before choosing the command; the contents of the clipboard will automatically appear in the dialog box that is opened.

Search Memory Dialog Box

Choosing the Search... (ALT, -, R) command opens the following dialog box:



Value	Lets you enter a value.
String	Lets you enter a string.
Start	Lets you enter the starting address of the memory range to search.
End	Lets you enter the end address of the memory range to search.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Searches for the specified value or string.

Close Closes the dialog box.

Command File Command

```
SEA(RCH) MEM(ORY) BYTE/WORD/LONG addr_range value

SEA(RCH) MEM(ORY) STR(ING) "string"
```

See Also

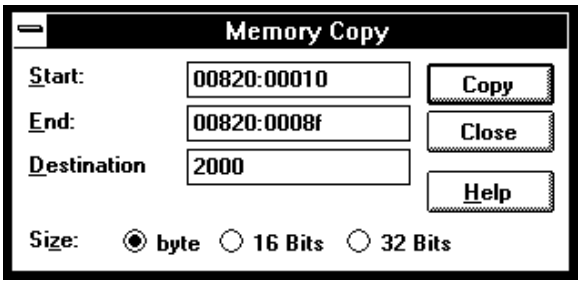
"To search memory for a value or string" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Copy... (ALT, -, U, C)

Copies the contents of one memory area to another.

Memory Copy Dialog Box

Choosing the Utilities→Copy... (ALT, -, U, C) command opens the following dialog box:



Start	Lets you enter the starting address of the source memory area.
End	Lets you enter the end address of the source memory area.
Destination	Specifies the starting address of the destination memory area.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Copies the memory contents.
Close	Closes the dialog box.

Command File Command

```
MEM(ORY) COP(Y) size address_range address
```

See Also

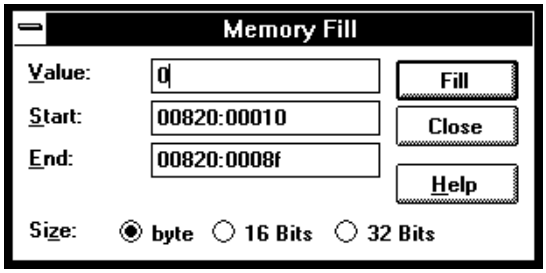
"To copy memory to a different location" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Fill... (ALT, -, U, F)

Fills a range of memory with a specified value.

Memory Fill Dialog Box

Choosing the Utilities→Fill... (ALT, -, U, F) command opens the following dialog box:



Value	Lets you enter the filling value.
Start	Lets you enter the starting address of the memory area to be filled.
End	Lets you enter the end address of the memory area to be filled.
Size	Selects the size of the filling value. If the value specified is larger than can fit in the size selected, the upper bits of the value are ignored. You can select the size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Executes the command.

Close Closes the dialog box.

Command File Command

```
MEM(ORY) FIL(L) size address_range data
```

See Also

"To modify a range of memory with a value" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Image... (ALT, -, U, I)

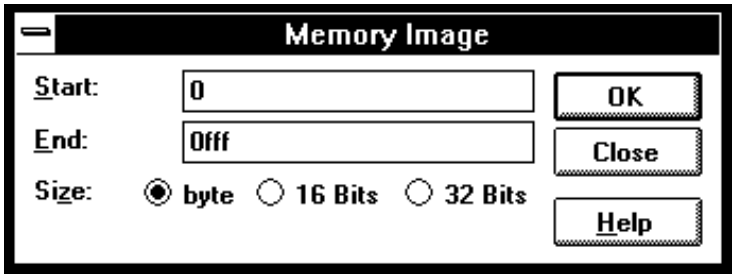
Copies the contents of a target system memory range into the corresponding emulation memory range.

You can copy programs that are in target system ROM to emulation memory. Once the program code is in emulation memory, you can use features like breakpoints, run until, etc.

The address range must be mapped as emulation memory before choosing this command.

Memory Image Dialog Box

Choosing the Utilities→Image... (ALT, -, U, I) command opens the following dialog box:



Start	Lets you enter the starting address of the memory area.
End	Lets you enter end address of the memory area.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Copies the target system memory into emulation memory.
Close	Closes the dialog box.

Command File Command

```
MEM(ORY) IMA(GE) size address_range
```

See Also

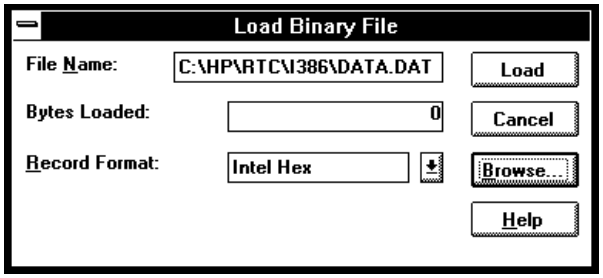
"To copy target system memory into emulation memory" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Load... (ALT, -, U, L)

Loads memory contents from a previously stored file.

Load Binary File Dialog Box

Choosing the Utilities→Load... (ALT, -, U, L) command opens the following dialog box:



File Name	Lets you enter the name of the file to load memory from.
Bytes Loaded	After you choose the Import button, this box shows the number of bytes that are loaded.
Record Format	Lets you specify the format of the file from which you're loading memory. You can load Motorola S-Record or Intel Hexadecimal format files.
Load	Starts the memory load.
Cancel	Closes the dialog box.
Browse...	Opens a file selection dialog box from which you can select the file name.

Command File Command

```
MEM(ORY) LOA(D) MOT(OSREC) filename

MEM(ORY) LOA(D) INT(ELHEX) filename
```

See Also

"To copy target system memory into emulation memory" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

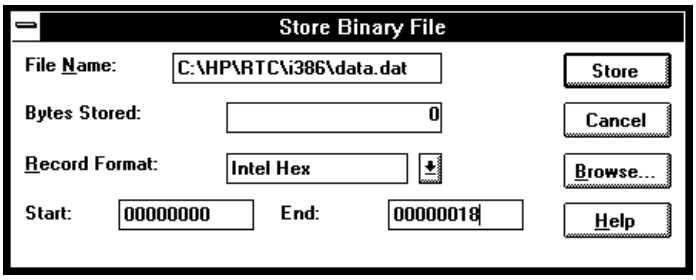
Utilities→Store... (ALT, -, U, S)

Utilities→Store... (ALT, -, U, S)

Stores memory contents to a binary file.

Store Binary File Dialog Box

Choosing the Utilities→Store... (ALT, -, U, S) command opens the following dialog box:



File Name	Lets you enter the name of the file to which memory contents are stored.
Bytes Stored	After you choose the Export button, this box shows the number of bytes that are stored.
Record Format	Lets you specify the format of the file to which you're storing memory. You can select Motorola S-Record or Intel Hexadecimal formats.
Start	Lets you enter the starting address of the memory range to be stored.

End	Lets you enter the ending address of the memory range to be stored.
Store	Starts the memory store.
Cancel	Closes the dialog box.
Browse...	Opens a file selection dialog box from which you can select a file name.

Command File Command

```
MEM(ORY) STO(RE) MOT(OSREC) addr-range filename

MEM(ORY) STO(RE) INT(ELHEX) addr-range filename
```

See Also

"To copy target system memory into emulation memory" in the "Displaying and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Load... (ALT, -, U, L)

GDT/LDT/IDT Window Commands

This section describes the following commands:

- Search→Entry... (ALT, -, R, E)
- Search→Selector... (ALT, -, R, S)

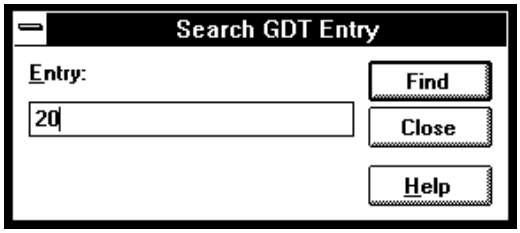
Search→Entry... (ALT, -, R, E)

Displays the specified entry in the window.

When the specified entry is found, it is displayed on the top line in the GDT, LDT, or IDT window. Choose the Window→GDT, Window→LDT, or Window→IDT command to open the window.

Search GDT/LDT/IDT Entry Dialog Box

Choosing the Search→Entry... (ALT, -, R, E) command opens a dialog box similar to the following:



The entry specifies the Nth entry in the table. For example, "20" specifies the twentieth entry line shown in the table. Because each entry is 8 bytes, the second entry starts at the 16th byte from the start of the table and the third entry starts at the 24th byte from the start of the table.

- Bits 15 through 3 of the selector specify the offset into the table of the start of the entry.

- Bit 2 specifies GDT when it is zero, or LDT when it is 1.
- Bits 1-0 specify privilege level. For example, if the second entry in the GDT is privilege level 0, its selector is 8. If it had a DPL of 3, it would be B (hex).

Find — Searches for the specified entry.

Close — Closes the dialog box.

Command File Command

```
GDT ENTRY value
```

```
LDT ENTRY value
```

See Also

"The GDT window", "The LDT window", or the "The IDT window" in the "Debugger Windows" section of the "Concepts" chapter.

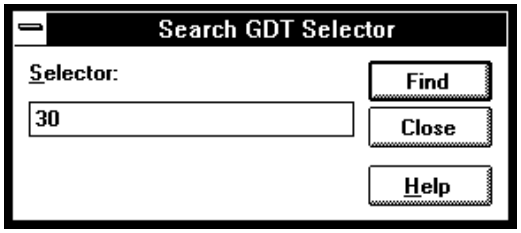
Search→Selector... (ALT, -, R, S)

Displays the specified selector in the window.

When the specified selector is found, it is displayed on the top line in the GDT, LDT, or IDT window. Choose the Window→GDT, Window→LDT, or Window→IDT command to open the window.

Search GDT/LDT/IDT Selector Dialog Box

Choosing the Search→Selector... (ALT, -, R, S) command opens a dialog box similar to the following:



To search for a selector, choose the Search→Selector... command. Then enter the selector number (in hex) and either press return or the Find button.

The lower three bits of the selector number are ignored on entry. For example, selector number 30 may be used to search for selector 30, 31, 32, or 33.

If the requested selector is within the range of the current table, it will be positioned at the top of the window. If it is out of range, an error box will pop up telling you it is an invalid selector.

Find	Searches for the specified selector.
Close	Closes the dialog box.

Command File Command

```
GDT SELECTOR value

LDT SELECTOR value
```

See Also

"The GDT window", "The LDT window", or the "The IDT window" in the "Debugger Windows" section of the "Concepts" chapter.

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)

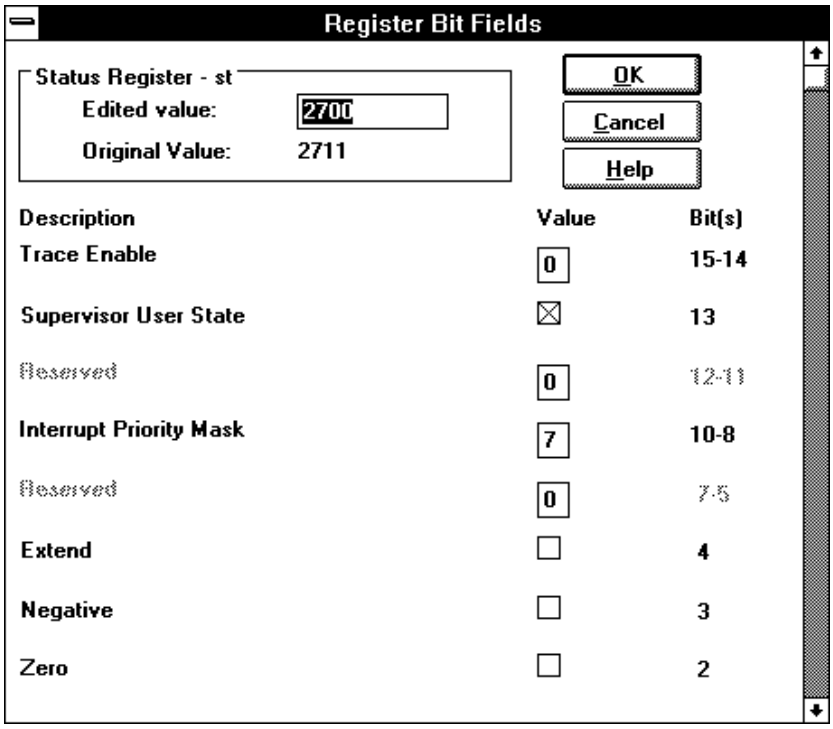
Copies the current Register window contents to the destination file specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
COP(Y) REG(ISTER)
```

Register Bit Fields Dialog Box

When a register has bit-fields, a dialog will pop-up and the register value may be edited by changing the whole value or by editing individual bit-fields.



When editing in the dialog box, a carriage-return is the same as choosing the OK button. To end an edit of a field within the dialog box without quitting, use the Tab key.

Edited Value — Shows the register value that corresponds to the selections made below. You can also change the register's value by modifying the value in this text box.

Original Value — Shows the value of the register when the dialog box was opened. If the register could not be read, 'XXXXXXXX' is displayed.

OK	Modifies the register as specified, and closes the dialog box.
Cancel	Closes the dialog box without modifying the register.

Source Window Commands

This section describes the following commands:

- Display→Mixed Mode (ALT, -, D, M)
- Display→Source Only (ALT, -, D, S)
- Display→Select Source... (ALT, -, D, L)
- Search→String... (ALT, -, R, S)
- Search→Function... (ALT, -, R, F)
- Search→Address... (ALT, -, R, A)

Display→Mixed Mode (ALT, -, D, M)

Chooses the source/mnemonic mixed display mode.

Command File Command

```
MOD(E) MNE(MONIC) ON
```

See Also

"To display source code mixed with assembly instructions" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Display→Source Only (ALT, -, D, S)

Chooses the source only display mode.

Command File Command

```
MOD(E) MNE(MONIC) OFF
```

See Also

"To display source code only" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

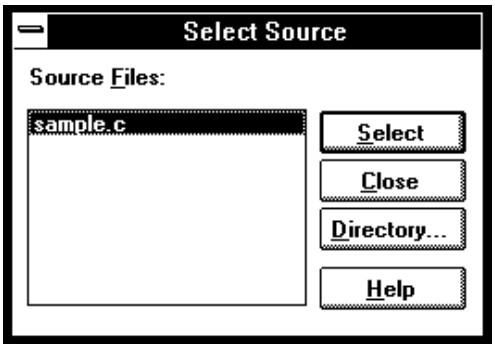
Display→Select Source... (ALT, -, D, L)

Displays the contents of the specified C source file in the Source window.

This command is disabled before the object file is loaded or when no source is available for the loaded object file.

Select Source Dialog Box

Choosing the Display→Select Source... (ALT, -, D, L) command opens the following dialog box:



Source Files	Lists C source files associated with the loaded object file. You can select the source file to be displayed from this list.
Select	Switches the Source window contents to the selected source file.
Close	Closes the dialog box.
Directory	Opens the Search Directories Dialog Box from which you can add directories to the search path.

Command File Command

```
FIL(E) SOU(RCE) module_name
```

See Also

"To display source files by their names" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Search→String... (ALT, -, R, S)

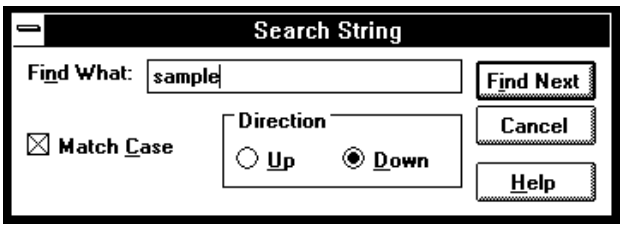
Searches for, and displays, a string in the Source window.

The search starts from the current cursor position in the Source window, may be either forward or backward, and may be case sensitive.

The string can be selected from another window (in other words, copied to the clipboard) before choosing the command; it will automatically appear in the dialog box that is opened.

Search String Dialog Box

Choosing the Search→String... (ALT, -, R, S) command opens the following dialog box:



Find What	Lets you enter the string.
Match Case	Selects or deselects case matching.
Up	Specifies that the search be from the current cursor position backward.
Down	Specifies that the search be from the current cursor position forward.

Find Next	Searches for the string.
Close	Closes the dialog box.

Command File Command

```
SEA(RCH) STR(ING) FOR/BACK ON/OFF strings
Searches the specified string in the specified
direction with the case matching option ON or OFF.
```

See Also

"To search for strings in the source files" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Search→Function... (ALT, -, R, F)

Searches for, and displays, a function in the Source window.

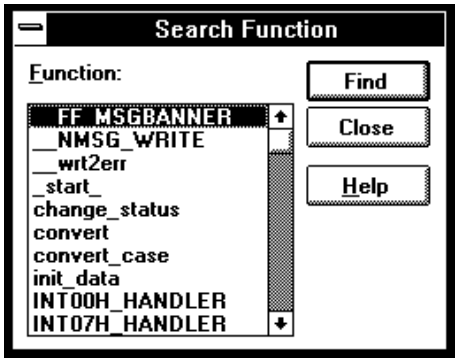
The object file and symbols must be loaded before you can choose this command.

Note

This command displays the source file based on the function information in the object file. Depending on the structure of the function, the command may fail in displaying the declaration of the function.

Search Function Dialog Box

Choosing the Search→Function... (ALT, -, R, F) command opens the following dialog box:



Function	Lets you select the function to search for.
Find	Searches the specified function.
Close	Closes the dialog box.

Command File Command

```
SEA(RCH) FUNC(TION) func_name
```

See Also

"To search for function names in the source files" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

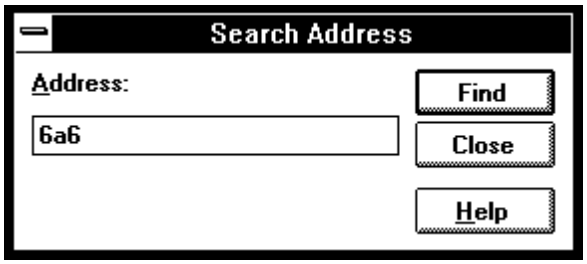
Search→Address... (ALT, -, R, A)

Searches for, and displays, an address in the Source window.

Address expressions such as function names or symbols can be selected from another window (in other words, copied to the clipboard) before choosing the command; the contents of the clipboard will automatically appear in the dialog box that is opened.

Search Address Dialog Box

Choosing the Search→Address... (ALT, -, R, A) command opens the following dialog box:



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

Command File Command

```
CUR(SOR) address
When used before the COME command, this command can be
used to run to a particular address.
```

See Also

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

Search Directories Dialog Box

Choosing the Directories... button in the Select Source dialog box opens the following dialog box:

Directory	Lets you enter the directory to be added to the source file search path.
Search Source Directories	Lists the directories in the source file search path.
Add	Adds the directory entered in the Directory text box to the source file search path.
Delete	Deletes the directory in the Directory text box from the source file search path.
Close	Closes the dialog box.

See Also

"To specify source file directories" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Symbol Window Commands

This section describes the following commands:

- Display→Modules (ALT, -, D, M)
- Display→Functions (ALT, -, D, F)
- Display→Externals (ALT, -, D, E)
- Display→Locals... (ALT, -, D, L)
- Display→Asm Globals (ALT, -, D, G)
- Display→Asm Locals... (ALT, -, D, A)
- Display→User defined (ALT, -, D, U)
- Copy→Window (ALT, -, P, W)
- Copy→All (ALT, -, P, A)
- FindString→String... (ALT, -, D, M)
- User defined→Add... (ALT, -, U, A)
- User defined→Delete (ALT, -, U, D)
- User defined→Delete All (ALT, -, U, L)

Display→Modules (ALT, -, D, M)

Displays the symbolic module information from the loaded object file.

Command File Command

```
SYM(BOL) LIS(T) MOD(ULE)
```

See Also

"To display program module information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Functions (ALT, -, D, F)

Displays the symbolic function information from the loaded object file.

The Symbol window displays the name, type and address range for C functions.

Command File Command

```
SYM(BOL) LIS(T) FUN(CTION)
```

See Also

"To display function information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Externals (ALT, -, D, E)

Displays the global variable information from the loaded object file.

The Symbol window displays the name, type and address for global variables.

Command File Command

```
SYM(BOL) LIS(T) EXT(ERNAL)
```

See Also

"To display external symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Locals... (ALT, -, D, L)

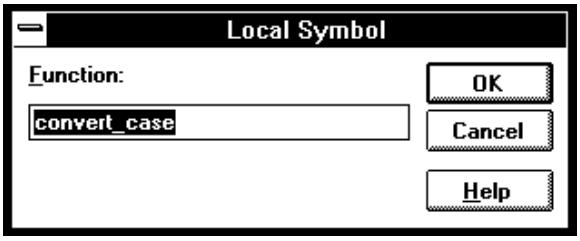
Displays the local variable information on the specified function.

The function name can be selected from another window (in other words, copied to the clipboard) before choosing the command; the clipboard contents automatically appear in the dialog box that is opened.

The Symbol window displays the name, type and offset from the frame pointer for the local variables for the specified function.

Local Symbol Dialog Box

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



Function	Selects the function for which the local variable information is displayed.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) LIS(T) INT(ERNAL) function
```

See Also

"To display local symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Asm Globals (ALT, -, D, G)

Displays the global Assembler symbol information from the loaded object file.

The Symbol window displays the name and address for the global assembler symbols.

Command File Command

```
SYM(BOL) LIS(T) GLO(BALS)
```

See Also

"To display global assembler symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→Asm Locals... (ALT, -, D, A)

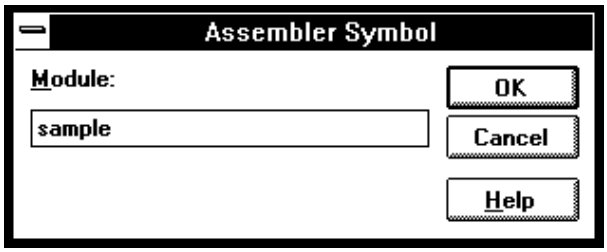
Displays the local symbol information from the specified module.

The module name can be selected from another window (in other words, copied to the clipboard) before choosing the command; the clipboard contents automatically appear in the dialog box that is opened.

The Symbol window displays the name and address for the local symbols for the specified module.

Assembler Symbol Dialog Box

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



Module	Selects the module for which the local symbols are displayed.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) LIS(T) LOC(AL) module
```

See Also

"To display local assembler symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Display→User defined (ALT, -, D, U)

Displays the user-defined symbol information.

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

The User defined→Add... (ALT, -, D, U) command adds the user-defined symbols.

Command File Command

```
SYM(BOL) LIS(T) USE(R)
```

See Also

"To display user-defined symbol information" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

Copy→Window (ALT, -, P, W)

Copies the information currently displayed in the Symbol window to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
SYM(BOL) COP(Y) DIS(PLAY)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Copy→All (ALT, -, P, A)

Copies all the symbol information to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
SYM(BOL) COP(Y) ALL
```

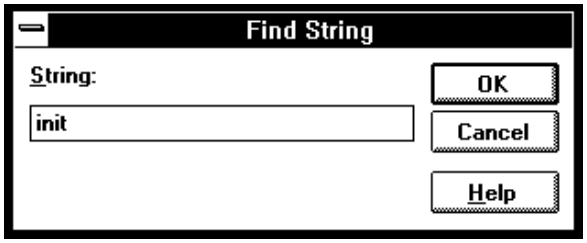
FindString→String... (ALT, -, F, S)

Displays the symbols that contain the specified string.

This command performs a case-sensitive search.

Symbol Matches Dialog Box

Choosing the FindString→String... (ALT, -, F, S) command opens the following dialog box:



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

Command File Command

```
SYM(BOL) MAT(CH) string
```

See Also

"To display the symbols containing the specified string" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

User defined→Add... (ALT, -, U, A)

Adds the specified user-defined symbol.

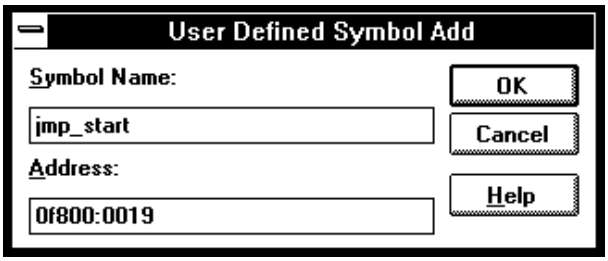
User-defined symbols may be used in debugger commands just like other program symbols.

The symbol name must satisfy the following requirements:

- The name must begin with an alphabetical, _ (underscore), or ? character.
- The following characters must be any of alphanumerical, _ (underscore), or ? characters.
- The maximum number of characters is 256.

User defined Symbol Dialog Box

Choosing the User defined→Add... (ALT, -, U, A) command opens the following dialog box:



Symbol Name	Specifies the symbol to be added.
Address	Specifies the address of the symbol.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
SYM(BOL) ADD symbol_nam address
```

See Also

"To create a user-defined symbol" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

User defined→Delete (ALT, -, U, D)

Deletes the specified user-defined symbol.

This command deletes the user-defined symbol selected in the Symbol window.

Command File Command

```
SYM(BOL) DEL(ETE) symbol_nam
```

See Also

"To delete a user-defined symbol" in the "Displaying Symbol Information" section of the "Debugging Programs" chapter.

User defined→Delete All (ALT, -, U, L)

Deletes all the user-defined symbols.

Command File Command

```
SYM(BOL) DEL(ETE) ALL
```

Trace Window Commands

This section describes the following commands:

- Display→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)
- Display→From State... (ALT, -, D, F)
- 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.

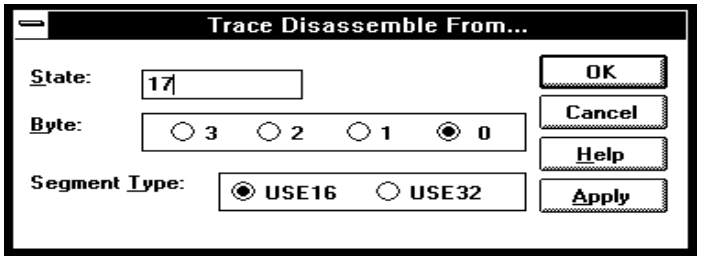
Trace Display→From State... (ALT -, D, F)

Lets you specify a state and/or a byte within a state where you wish to begin disassembly, as well as a segment type (16-bit or 32-bit).

Normally the disassembler begins disassembly with the first byte of the first captured op-code fetch. Sometimes this does not result in correct disassembly because the first byte is a continuation of a previous opcode. When a branch-trace message is found, the disassembler will re-synchronize. However, this dialog box allows you to manually set the correct starting byte.

Trace Disassemble From... Dialog Box

Choosing the Display→From State... (ALT, -, D, F) command opens the following dialog box:



State — Lets you enter a state number (as shown in the left-most column in the trace display) where you wish to begin disassembly.

Byte — Lets you specify the byte within the selected state where you wish to begin disassembly.

Segment Type — Lets you specify what type of segment (16-bit or 32-bit) the code is in. You may specify this without specifying a disassembly state.

Note that the state you specify should be a control read (instead of a data read).

Command File Command

```
MODE TRACE DISPLAY FROM <state>

MODE TRACE DISPLAY BYTE0/BYTE1/BYTE2/BYTE3

MODE TRACE DISPLAY USE16/USE32
```

Copy→Window (ALT, -, P, W)

Copies the information currently in the Trace window to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
TRA(CE) COP(Y) DIS(PLAY)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.

Copy→All (ALT, -, P, A)

Copies all the trace information to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
TRA(CE) COP(Y) ALL
```

Search→Trigger (ALT, -, R, T)

Positions the trigger state at the top of the Trace window.

Command File Command

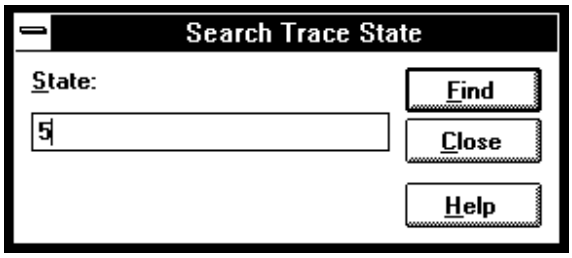
```
TRA(CE) FIN(D) TRI(GGER)
```

Search→State... (ALT, -, R, S)

Positions the specified state at the top of the Trace window.

Search Trace State Dialog Box

Choosing the Search→State... (ALT, -, R, S) command opens the following dialog box:



State	Lets you enter the trace state number to search for.
Find	Searches for the specified trace state.
Close	Closes the dialog box.

Command File Command

```
TRA(CE) FIN(D) STA(TE) state_num
```

Trace Spec Copy→Specification (ALT, -, T, S)

Copies the current trace specification to the listing file.

Command File Command

```
TRA(CE) COP(Y) SPE(C)
```

Trace Spec Copy→Destination... (ALT, -, T, D)

Names the listing file to which debugger information may be copied.

This command opens a file selection dialog box from which you can select the listing file. Listing files have the extension ".LST".

Command File Command

```
COP(Y) TO filename
```

WatchPoint Window Commands

This section describes the following command:

- Edit...

Edit... (ALT, -, E)

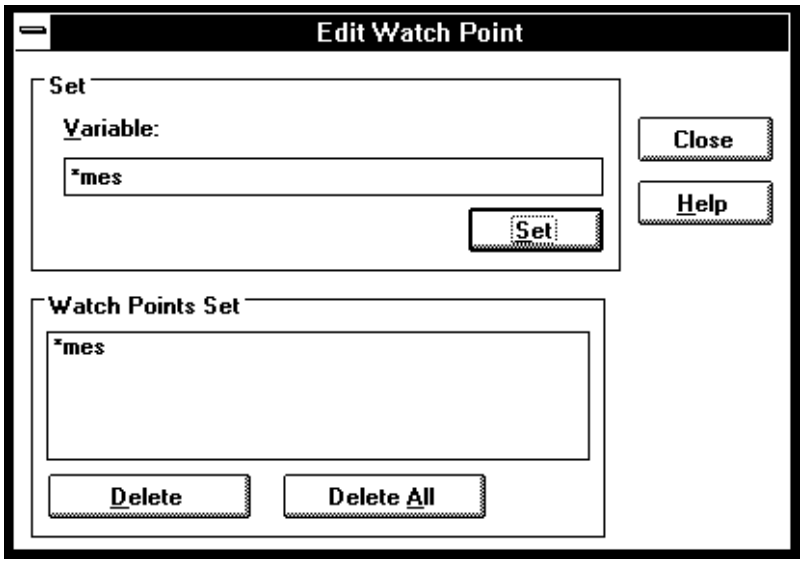
Registers or deletes watchpoints.

Variables can be selected from another window (in other words, copied to the clipboard) before choosing the Edit... (ALT, -, E) command from the WatchPoint window's control menu, and they will automatically appear in the dialog box that is opened.

Dynamic variables can be registered and displayed in the WatchPoint window when the current program counter is in the function in which the variable is declared. If the current program counter is not in the function, the variable name is invalid and results in an error.

WatchPoint Dialog Box

Choosing the Edit... (ALT, -, E) command from the WatchPoint window's control menu opens the following dialog box:



Variable	Lets you enter the name of the variable to be registered as a watchpoint. The contents of the clipboard, usually a variable selected from another window, automatically appears in this text box.
Watch Points Set	Lists the current watchpoints and allows you to select the watchpoint to be deleted.
Set	Copies the specified variable to the WatchPoint window.
Delete	Deletes the variable selected in the Watch Points Set box.
Delete All	Deletes all the watchpoints.
Close	Closes the dialog box.

Command File Command

```
WP SET address
Registers the specified address as a watchpoint.

WP DEL(ETE) address
Deletes the specified watchpoint.

WP DEL(ETE) ALL
Deletes all the current watchpoints.
```

See Also

"To monitor a variable in the WatchPoint window" in the "Displaying and Editing Variables" section of the "Debugging Programs" chapter.

"Symbols" in the "Expressions in Commands" chapter.

10

Window Pop-up Menu Commands

Window Pop-up Menu Commands

This chapter describes the commands that can be chosen from the pop-up menus in debugger windows. Pop-up menus are accessed by clicking the right mouse button in the window.

- BackTrace Window Pop-up Commands
- Source Window Pop-up Commands

BackTrace Window Pop-up Commands

- Source at Stack Level

Source at Stack Level

For the cursor-selected function in the BackTrace window, this command displays the function call in the Source window.

Source Window Pop-up Commands

- Set Breakpoint
- Clear Breakpoint
- Evaluate It
- Add to Watch
- Run to Cursor

Set Breakpoint

Sets a breakpoint on the line containing the cursor. Refer to the Breakpoint→Set at Cursor (ALT, B, S) command.

Clear Breakpoint

Deletes the breakpoint on the line containing the cursor. Refer to the Breakpoint→Delete at Cursor (ALT, B, D) command.

Evaluate It

Evaluates the clipboard contents and places the result in the Expression window. Refer to the Evaluate... (ALT, -, E) command available from the Expression window's control menu.

Add to Watch

Adds the selected variable (that is, the variable copied to the clipboard) to the WatchPoint window. Refer to the Variable→Edit... (ALT, V, E) command.

Run to Cursor

Executes the program up to the Source window line containing the cursor. Refer to the Execution→Run to Cursor (ALT, R C) command.

11

Other Command File and Macro Commands

Other Command File and Macro Commands

This chapter describes the commands that are only available in command files, break macros, or buttons.

- BEEP
- EXIT
- FILE CHAINCMD
- FILE RERUN
- NOP
- TERMCOM
- WAIT

BEEP

Sounds beep during command file or break macro execution.

Command File Command

BEEP

EXIT

Exits, or conditionally exits, command file execution.

Command File Command

```
EXIT
Exits command file execution.

EXIT VAR(IABLE) address value
Exits command file execution if the variable contains
the value.

EXIT REG(ISTER) regname value
Exits command file execution if the register contains
the value.

EXIT MEM(ORY) BYTE/WORD/LONG address value
Exits command file execution if the memory location
contains the value.

EXIT IO BYTE/WORD address value
Exits command file execution if the I/O location
contains the value.
```

FILE CHAINCMD

Chains command file execution.

This command lets you run one command file from another nonrecursively; in other words, control is not returned to the original command file.

By contrast, the FILE COMMAND command is recursive; if you use the FILE COMMAND command to run one command file from another, control will be returned to the original command file. FILE COMMAND commands can be nested four levels deep.

Command File Command

```
FILE CHAINCMD filename
```

FILE RERUN

Starts command file execution over again.

This command is useful for looping stimulus files or running a demo or other command file continuously.

Command File Command

```
FILE RERUN
```

NOP

No operation.

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

Command File Command

```
NOP
```

```
NOP comments
```

TERMCOM

Sends Terminal Interface commands to the HP 64700.

You cannot see output from these commands. Refer to your Installation/Service/Terminal Interface User's Guide for more information about these commands.

Command File Command

```
TERMCOM ti-command
```

WAIT

Inserts wait delays during command file execution.

Command File Command

```
WAI(T) MON(ITOR)
Waits until MONITOR status.

WAI(T) RUN
Waits until RUN status.

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

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

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

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

WAI(T) GRA(NT)
Waits until BUS GRANT status

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

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

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

WAI(T) TIM(E) seconds
Waits for a number of seconds.
```

12

Error Messages

Error Messages

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, Windows for Workgroups)

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 \LANMAN.DOS\ETC\HOSTS (HP-ARPA) or \WINDOWS\HOSTS (Windows for Workgroups) may contain entries of the form:

```
system1  15.6.28.0
```

Note The directory of the "hosts" file may be different on your system.

If "HP Probe" or "DNR" (Domain Name Resolution) is available on your PC, those are consulted first for a mapping between the hostname and the IP address. If the hostname is not found by that method, or if those services are unavailable, the local "hosts" file is consulted for the mapping.

Note that if "Probe" is available on your system but unable to resolve the address, there will be 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 NET.CFG files are searched.

Incorrect LAN Address (WINSOCK)

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

An IP address consists of 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 hosts file may contain entries of the form:

```
system1  15.6.28.0
```

Note

Because WINSOCK is a standard interface to many LAN software vendors, you need to read your LAN vendor's documentation before specifying the LAN address.

Internal error in communications driver

These types of errors typically occur because other applications have used up a limited amount of some kind of global resource (such as memory or sockets).

You usually have to reboot the PC to free the global resources used by the communications driver.

Internal error in Windows

These types of errors typically occur because other applications have used up a limited 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 Enter 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 recreate 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 reinstall the debugger software.

Out of DOS Memory for LAN buffer

This means that there is not enough memory in the lower 1 Mbyte of address space (that is, conventional memory) for the LAN driver to allocate a buffer to communicate with the LAN TSR.

When you are in windows, and execute the DOS command "mem," you cannot see the memory that is in the lower 1 Mbyte that is used by the windows program. If you have the Microsoft program "heapwalker," you can use it to see what programs have allocated space in the address range 0 through FFFFF.

To fix this, you can:

- Reduce the number of TSRs running on your PC (before Windows starts) that use conventional memory.
- Reconfigure your network to have fewer sockets or modules loaded, or to be configured for fewer total connections.
- Use a different memory manager to reduce your network memory usage, such as QEMM.

Out of Windows timer resources

The debugger is not able to acquire the timer resources it needs.

There are a limited number of timer resources in Windows. You may be able to free timer resources by closing other applications.

PC is out of RAM memory

The debugger is not able to acquire the memory it needs because other applications are using it or because of fragmented memory.

You may be able to free memory by closing other applications, or you might have to reboot the PC to cause memory to be unfragmented.

Timed out during communications

The HP 64700 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.

```
[TCPIP_XFR]
TCPCONNTIMEOUT=<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
- Monitor Program Options
- Trace Signals and Predefined Status Values
- Understanding 80386 Analysis
- Understanding Address, Data, and Status
- Entering Addresses as Constants
- Unexpected Stepping Behavior

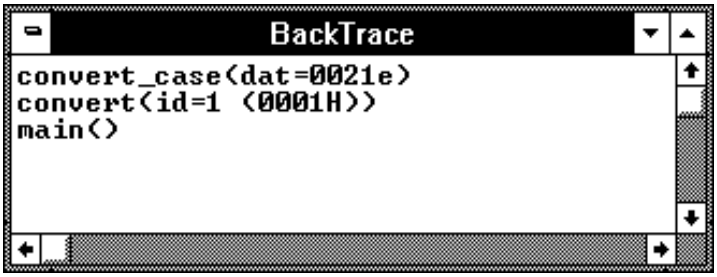
Debugger Windows

This section describes the following debugger windows:

- BackTrace
- Button
- Expression
- I/O
- Memory
- GDT
- LDT
- IDT
- Register
- Source
- Status
- Symbol
- Trace
- WatchPoint

The BackTrace Window

The BackTrace window displays the function associated with the current program counter value and this function's caller functions 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.

Note that the return address can occur any number of bytes from the base pointer of the stack. The OMF386 symbol file contains information used to locate return addresses. If symbols are not available (typically for assembly-language routines), the backtrace is shown as far as it can decode the addresses, and then display of the backtrace stops.

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 pop-up menu command. Cursor-select a function in the BackTrace window and choose this command to display (in the Source window) the code that called the function.

See Also

"BackTrace Window Pop-Up Commands" in the "Window Pop-Up Commands" chapter.

The Button Window

The Button window contains user-defined buttons that, when chosen, execute debugger commands or command files.

The Button window's *control menu* provides the Edit... (ALT, -, E) command which lets you add and delete buttons from the window.

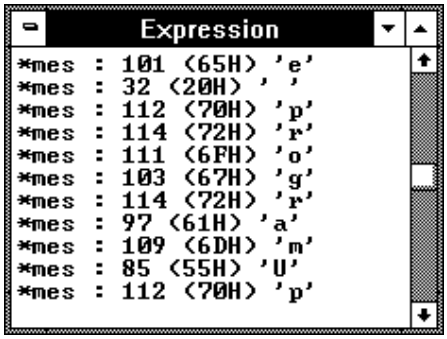
See Also

"Using Command Files" in the "Using the Debugger Interface" chapter.

"Button Window Commands" in the "Window Control Menu Commands" chapter.

The Expression Window

The Expression window displays the results of the EVALUATE commands in command files or break macros.



When a variable name is specified with the EVALUATE command, the Expression window displays the evaluation of the variable. When a quoted string of ASCII characters is specified with the EVALUATE command, the Expression window displays the string.

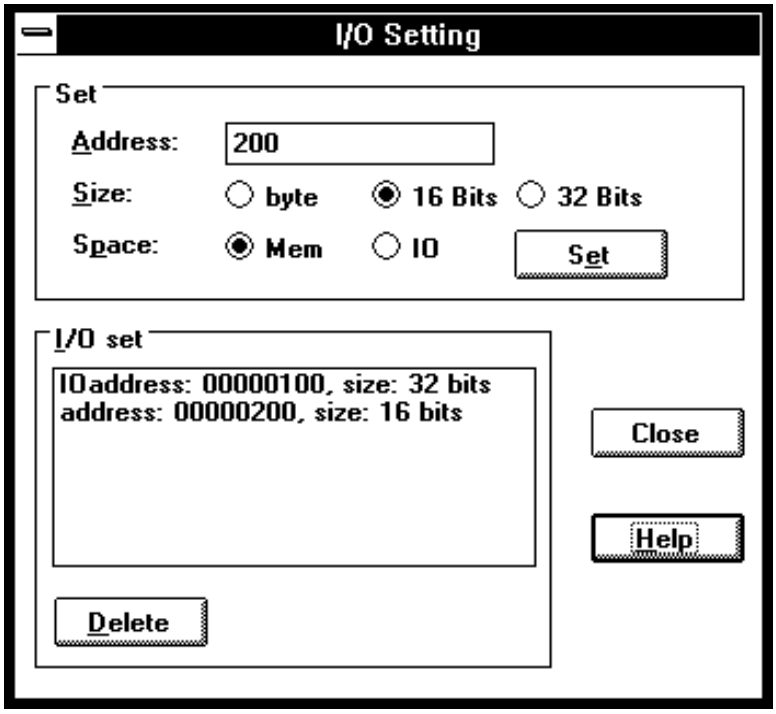
The Expression window's *control menu* provides the Evaluate... (ALT, -, E) command which lets you evaluate expressions and see the results in the window.

See Also

"Expression Window Commands" in the "Window Control Menu Commands" chapter.

The I/O Window

The I/O window displays the contents of the I/O locations.



You can modify the contents of I/O locations by double-clicking on the value, using the keyboard to type in the new value, and pressing the Enter key.

The I/O window contents are updated periodically when the processor is running the user program.

If a location is in target system memory, a temporary break from the user program into the monitor program must occur in order for the debugger to update or modify that location's contents. If it's important that the user program execute without these types of interruptions, you should disallow monitor intrusion. Even when monitor intrusion is allowed, you can stop temporary breaks during the window update by turning polling OFF.

Note that if any address in the displayed range is not readable (for example, it is beyond the segment limit in protected mode), all memory will be displayed as dashes (--). In this case, resize the memory window to only display the address ranges needed.

Also, do not use the memory window for reading memory-mapped I/O devices; use the I/O window (to ensure that only the bytes necessary to read the specific address are read).

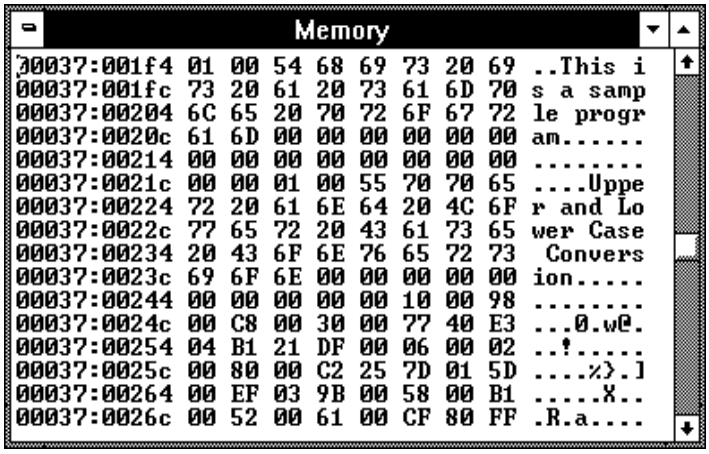
See Also

"Displaying and Editing I/O Locations" in the "Debugging Programs" chapter.

"I/O Window Commands" in the "Window Control Menu Commands" chapter.

The Memory Window

The Memory window displays memory contents.



The Memory window has *control menu* commands that let you change the format of the memory display and the size of the locations displayed or modified. When the absolute (single-column) format is chosen, symbols corresponding to addresses are displayed. When data is displayed in byte format, ASCII characters for the byte values are also displayed.

When Memory window polling is turned ON, you can modify the addresses displayed or contents of memory locations by double-clicking on the address or value, using the keyboard to type in the new address or value, and pressing the Enter key.

The Memory window contents are updated periodically when the processor is running the user program.

If a location is in target system memory, a temporary break from the user program into the monitor program must occur in order for the debugger to update or modify that location's contents. If it's important that the user program execute without these types of interruptions, you should disallow monitor intrusion. Even when monitor intrusion is allowed, you can stop temporary breaks during the window update by turning polling OFF.

In contrast to the memory window, the I/O window only reads the number of bytes specified in the Size field when it displays the data. The memory window reads a buffer which may contain many more bytes than are displayed. Therefore, if a memory address is surrounded by addresses you do not want to read, use the I/O window to avoid reading the surrounding addresses. Typically, you will want to use the I/O window when displaying memory-mapped I/O.

See Also

"Displaying and Editing Memory" in the "Debugging Programs" chapter.

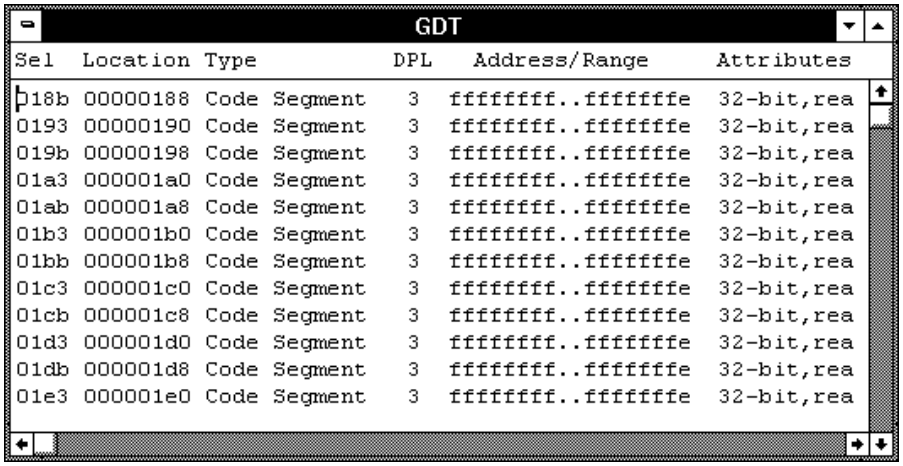
"Memory Window Commands" in the "Window Control Menu Commands" chapter.

The GDT Window

The GDT window displays the contents of the current Global Descriptor Table. The current GDT can be found by looking at the current value of the gdtr.b (GDT base) and gdtr.l (GDT limit) registers in the System Registers window.

You cannot display the GDT window (or gdtr.l and gdtr.b) if the emulator is running your target program and monitor intrusion is disallowed unless the GDT is in dual-port memory.

If you are in real-mode (prior to entering protected mode), you cannot display a valid GDT window until the LGDT opcode has been executed, or you have modified the gdtr.b register.



Note that selector 0 is always the NULL selector. Referencing it in an 80386 program will always cause the 80386 to generate a General Protection Fault.

Each display line has six fields:

- Sel. The selector of the segment. This is the value loaded into a segment descriptor (CS, DS, etc). The last two bits of the selector are the DPL of the segment.
- Location. This is the physical address of this entry. This is useful when looking at trace lists.

- Type. This decodes the type of the segment. Types include LDT (this entry points to a Local Descriptor Table), Code segments, Data segments, TSS blocks, and various gates.
  "80286 call gates/TSS/etc" are decoded simply as the type, but the attribute will include "16-bit".
  "80386 gates/TSS/etc" are decoded simply as the type, but the attribute will be "32-bit".
- DPL. This is the Descriptor Privilege level of the entry.
- Address/Range. This is either a starting and ending address for the entry or a selector (depending on the type of entry). For expand-down segments, the address range is the real address range (that is, the wrapping is taken into account). For example, if the mapping file shows the range as 'start=00001EE4H limit=FFFFEFFFH Expand-down', the Address/Range column will show the range as 'ee4..1ee3', which is the linear address range that will be used.
- Attributes. This decodes the attributes according to the type of entry.

The GDT window shows the descriptor table in memory, not the shadow registers in the CPU. In order to change the shadow registers in the CPU, you must change the GDT table in memory (using the memory window). Break into the monitor, modify the desired segment register, and then exit the monitor.

See Also

"Searching for Entries", and "Searching for Selectors" in the "GDT/LDT/IDT Window Commands" section of the "Debugging Programs" chapter.

The LDT Window

The LDT window displays the contents of the current Local Descriptor Table. (The current LDT can be found by looking at the current value of the ldtr.b and ldtr.l (LDT base and limit) registers in the System Registers window.)

You cannot display the LDT window if the emulator is running your target program with monitor intrusion disallowed unless the LDT is in dual-port memory.

If you are in real-mode (prior to entering protected mode), you cannot display valid LDT window content until the LLDT opcode has been executed, or you have modified the ldtr.b register.

The selector numbers have bit 2 set (that is, the first selector is 4, not 0). That is how the processor differentiates between a selector in the GDT and the same selector in the LDT. Selector 4 (entry 0) is legal, but some builders leave it empty.

See the GDT window for descriptions of each of the six fields in the display lines.

The LDT window shows the descriptor table in memory, not the shadow registers in the CPU. In order to change the shadow registers in the CPU, you must change the LDT table in memory (using the memory window). Break into the monitor, modify the desired segment register, and then exit the monitor.

See Also

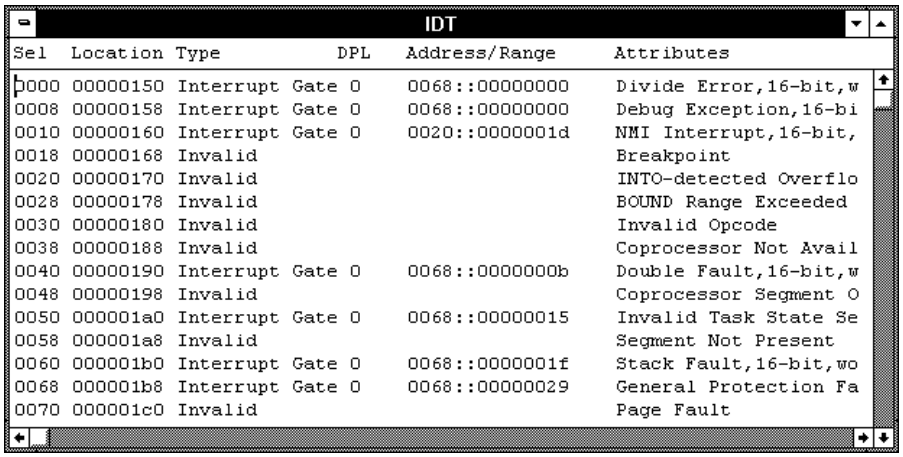
"Searching for Entries", and "Searching for Selectors" in the "GDT/LDT/IDT Window Commands" section of the "Debugging Programs" chapter.

The IDT Window

The IDT window displays the contents of the current Interrupt Descriptor Table. The current IDT can be found by looking at the current value of the idtr register in the System Registers window. It refers to an entry in the GDT, which in turn points to the linear address of the table.

You cannot display the IDT window if the emulator is running your target program and monitor intrusion is disallowed unless the IDT is in dual-port memory.

The IDT window display is only useful in protected mode.



- Sel. The selector of the descriptor.
- Location. This is the physical address of the entry. This is useful when looking at trace lists.
- Type. This decodes the type of the selector. Only interrupt gates, trap gates, and task gates may be in the IDT.
- DPL. This is the Descriptor Privilege level of the entry.
- Address/Range. This is the address of the interrupt routine or task TSS.
- Attributes. This decodes the attributes according to the type of entry.

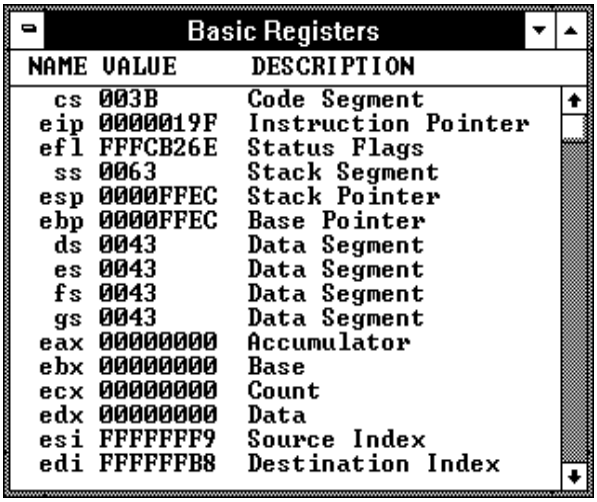
The name of the interrupt is displayed for the first 32 entries.

See Also

"Searching for Entries", and "Searching for Selectors" in the "GDT/LDT/IDT Window Commands" section of the "Debugging Programs" chapter.

The Register Windows

The Register windows display the contents of registers. There is a separate window for each class of registers. For example, the Basic Registers are in one class of 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 Register window contents are updated periodically when the processor is running the user program and monitor intrusion is allowed.

A temporary break from the user program into the monitor program must occur in order for the debugger to update or modify register contents. If it is important that the user program execute without these types of interruptions, you should disallow monitor intrusion.

See Also

"Displaying and Editing Registers" in the "Debugging Programs" chapter.

"Register Window Commands" in the "Window Control Menu Commands" chapter.

The Source Window

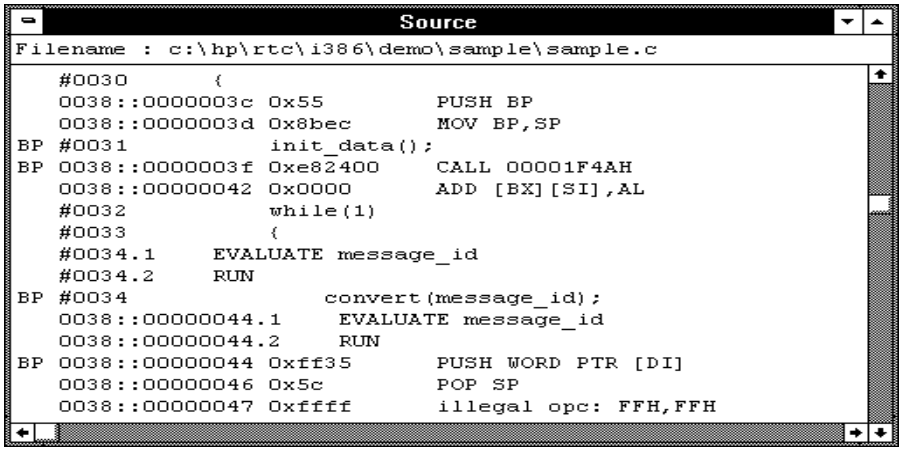
The Source window displays source files, optionally with disassembled instructions intermixed.

The Source window contains a cursor whose position is used when setting or deleting breakpoints or break macros or when running the program up to a certain line.

The Source window lets you copy strings, usually variable or function names to be used in commands, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

The Source window also provides commands in the *control menu* that let you select whether disassembled instruction mnemonics should appear intermixed with the C source code.

By clicking the right mouse button in the Source window, you can also access popup menu commands.



Filename	The name of the displayed source file appears at the top of the window.
Source Lines	C source code is displayed when available. Source lines are preceded by the corresponding line numbers. When programs are written in assembly language or when no C source code is available, disassembled instruction mnemonics are displayed.
Disassembled Instructions	In the Mnemonic Display mode, disassembled instruction mnemonics are intermixed with the source lines. Disassembled lines contain address, data, and mnemonic information. When symbolic information is available for the address, the corresponding symbol line precedes the disassembled instruction, displayed in the module_name\\symbol_name format.
Current PC	The line associated with the current program counter is highlighted.

Scroll Bars	For C source files, the display scrolls within the source files. For assembly language programs or programs for which no source code is available, the display scrolls for all the memory space.
"BP" Marker	The breakpoint marker, "BP", appears at the beginning of the breakpoint lines or break macro lines.
Break Macro Lines	Decimal points following line numbers or addresses indicate break macro lines.

Note

When programs are stored in target system memory and the emulator is running in real-time, source code cannot be displayed.

See Also

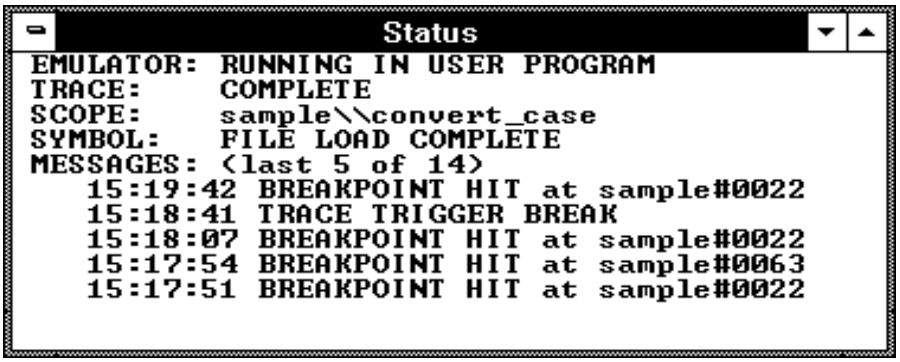
"Loading and Displaying Programs", "Stepping, Running, and Stopping the Program", and "Using Breakpoints and Break Macros" in the "Debugging Programs" chapter.

"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.

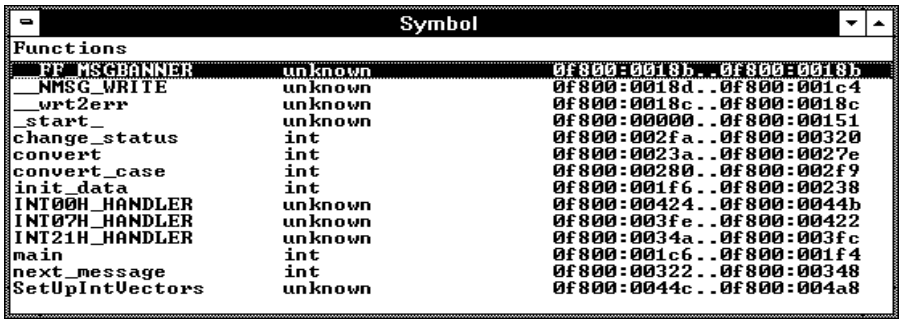
The Symbol Window

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 lets you display various types of symbols, add or delete user-defined symbols, copy Symbol window information, or search for symbols that contain a particular string.

The Symbol window lets you copy symbols to the clipboard by clicking the left mouse button. The symbol information can then be pasted from the clipboard in other commands.



Symbols are displayed with "type" and "address" values where appropriate.

See Also

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

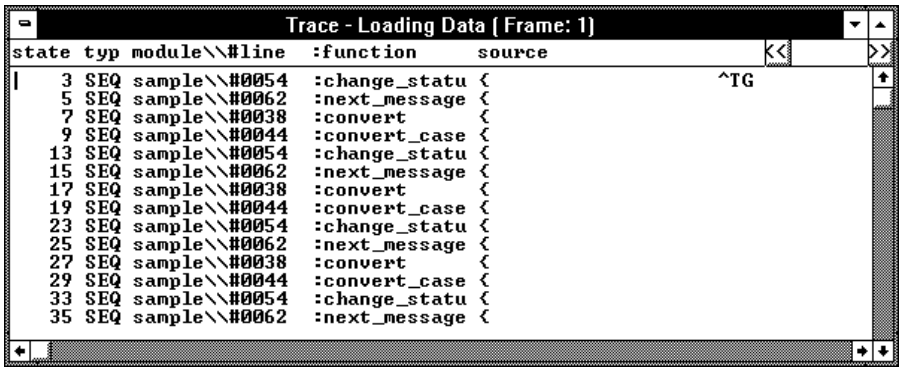
"Symbol Window Commands" in the "Window Control Menu Commands" chapter.

The Trace Window

The Trace window displays trace results and shows source code lines that correspond to the execution captured by the analyzer. Optionally, bus cycle states can be displayed along with the source code lines.

The Trace window has *control menu* commands that let you display bus cycles, specify whether count information should be 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 WatchPoint Window

The WatchPoint window displays the contents of variables that have been registered with the Variable→Edit... (ALT, V, E) command or with the Edit... (ALT, -, E) command in the WatchPoint window's control menu.

The contents of dynamic variables are displayed only when the current program counter is in the function in which the variable is declared.

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

The WatchPoint window lets you copy text strings, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

See Also

"Displaying and Editing Variables" in the "Debugging Programs" chapter.

"WatchPoint Window Commands" in the "Window Control Menu Commands" chapter.

Monitor Program Options

- Background monitor
- Foreground monitor
- Foreground monitor advantages and disadvantages

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 (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 memory. After the monitor has performed its task, emulation processor execution returns to the area where it was executing before the break.

The emulation monitor program executes out of a separate, internal memory system known as background memory, which is dual ported. A monitor program executing out of background memory is known as a background monitor program.

The foreground emulation monitor program also executes out of dual-port memory, which is not the same 8K, dual-port memory available to your programs. However, the foreground monitor does consume memory address space (that is, you must reserve physical addresses to contain the foreground monitor), and addresses consumed by the foreground monitor are not available to use within your target system.

Emulator firmware includes both background and foreground monitor programs and lets you select either one. You can also load and use a customized foreground monitor program, if desired.

Background monitor

The default emulator configuration selects the background monitor.

Interrupts from the target system are disabled during background monitor execution. If your programs have strict real-time requirements for servicing target system interrupts, you must use a foreground monitor.

DMA cycles are allowed while in the background monitor (that is, the HOLD line will be acknowledged with the HLDA signal even while executing the background monitor.)

Foreground monitor

A foreground monitor source file is provided with the emulator. It can be assembled, linked, and loaded into the debugger. It is linked and loaded separately from your program. However, you must provide:

- A physical address space of 16K that is not used for any other purpose within your target hardware.
- An unused entry in your GDT. You do not need to put any data in this entry. It will be filled in by the emulator prior to entering the monitor.
- If you are using paging, the foreground monitor must be located in address space where each virtual address is the same as each physical address (virtual address = physical address). You must have a valid page table for the virtual address range (although the specific entries for the foreground monitor will be filled in by the emulator prior to entering the monitor).

Foreground monitor advantages and disadvantages

Advantages

- A foreground monitor executes as part of the user program. This allows you to enable target system interrupts during monitor program execution for applications that have strict real-time processing requirements.
- A foreground monitor can be customized.

Disadvantages

- A foreground monitor consumes target system address space.
- In order for interrupts to be received while execution is in the monitor, they must either have a DPL of 0 (because the monitor runs at DPL 0), or be a task gate.
- A foreground monitor does not require target system stack space. However, because the foreground monitor runs at DPL 0, you must provide a privilege level 0 stack in case interrupts are serviced while the foreground monitor is executing.

Trace Signals and Predefined Status Values

This section describes how emulation-bus analyzer trace signals are assigned to microprocessor address bus, data bus, and control signals.

See also "Understanding Address, Data, and Status", and "Understanding 80386 Analysis" for more information.

Emulation-bus Analyzer Trace Signals

Trace Signals	Signal Name	Signal Description
0-31	A0-A31	Address Lines A2-A31, plus addresses 0-1 derived from BE0#-BE3#
32-63	D0-D31	Data lines 0-31
64	Monitor/User	0 = Monitor, 1 = User program execution
65	W/R#	1 = write, 0 = read
66	D/C#	1 = data, 0 = control
67	M/IO#	1 = memory, 0 = I/O
68	BE0#	0 = byte 0 (data bits 0-7) enabled
69	BE1#	0 = byte 1 (data bits 8-15) enabled
70	BE2#	0 = byte 2 (data bits 16-23) enabled
71	BE3#	0 = byte 3 (data bits 24-31) enabled
72	BS16#	0 = Bus Size 16 is asserted. Ignored for emulation memory accesses
73	NA#	0 = Next Address (pipelining) requested
74	LOCK#	0 = lock asserted (HOLD will not be acknowledged)
75	PEREQ	1 = Coprocessor has data to be transferred to the 80386
76	BUSY#	0 = Coprocessor is busy
77	ERROR#	0 = Coprocessor error
78	INTR	1 = Interrupt Request
79	HLDA	1 = Hold Acknowledge in previous cycle

Predefined Status Values

Qualifier	Status Bits (31-16)	Description
be0	xxxx xxxx xxx0 xxxx	BE0# (Byte Enable 0) active
be1	xxxx xxxx xx0x xxxx	BE1# (Byte Enable 1) active
be2	xxxx xxxx x0xx xxxx	BE2# (Byte Enable 2) active
be3	xxxx xxxx 0xxx xxxx	BE3# (Byte Enable 3) active
bs16	xxxx xxx0 xxxx xxxx	BS16# (Bus Size 16) active
btmsg	xxxx x1xx xxxx 001x	Branch Trace Message
busy	xxx0 xxxx xxxx xxxx	BUSY# (from the coprocessor) active
ctrl	xxxx xxxx xxxx x0xx	A control access (op-code fetch, for example)
data	xxxx xxxx xxxx x1xx	A data access (memory read, for example)
error	xx0x xxxx xxxx xxxx	ERROR# (from the coprocessor) active
halt	xxxx xxxx 1011 101x	The ’hlt’ instruction was executed
hlda	1xxx xxxx xxxx xxxx	HLDA (hold acknowledge) was active just prior to captured state (a DMA occurred)
inta	xxxx x0xx 1110 000x	An interrupt acknowledge cycle
intr	x1xx xxxx xxxx xxxx	INTR (interrupt request) line is active
io	xxxx xxxx xxxx 01xx	An I/O access (’out’, for example)
iord	xxxx xxxx xxxx 010x	An I/O read cycle
iowr	xxxx xxxx xxxx 011x	An I/O write cycle
lock	xxxx x0xx xxxx xxxx	LOCK# (locked cycle)
mem	xxxx xxxx xxxx 1xxx	A memory access (’read’, for example)
memif	xxxx xxxx xxxx 100x	A memory instruction fetch (op-code fetch)
memrd	xxxx xxxx xxxx 1x0x	A memory read
memwr	xxxx xxxx xxxx 111x	A memory write
mon	xxxx xxxx xxxx xxx0	A background monitor cycle
na	xxxx xx0x xxxx xxxx	NA# (pipelining) request active
pereq	xxxx 1xxx xxxx xxxx	PEREQ (from the coprocessor) active
read	xxxx xxxx xxxx xx0x	A read cycle (memory or I/O)
shut	xxxx xxxx 1110 101x	Processor shutdown
ttmsg	xxxx x1xx xxxx 000x	Task Trace message
write	xxxx xxxx xxxx xx1x	A write cycle (memory or I/O)

Understanding 80386 Analysis

The external address, data, and control signals of the 80386 can be difficult to understand. This section will help you understand how the 80386 works, how to interpret the trace information, and how to ask for more precise trace information.

Instruction reads are always four bytes

The 80386 always reads four bytes at a time when reading instructions. This can be confusing when the target of a branch is at an address that is not a multiple of four. This can also cause problems when you want to trigger on a specific function. See Understanding Address, Data, and Status for information on how the emulator helps you do this.

Prefetching

The 80386 may read up to 12 bytes of data before it starts to execute the first byte of data. Eleven of these twelve bytes of data are "prefetched" (that is, fetched from memory before they are needed). One implication of these prefetches is that the processor runs faster. Another is that the order of the external bus cycles can be confusing when you see them in a trace list.

Consider the following assembly code:

```
103E                readloop:
103E A10000             mov ax,control
1041 3D0000             cmp ax,0
1044 74F8               je readloop

1046                try1:
1046 3D0100             cmp ax,1
1049 75189090           jne try2 ;  command 1: call into ldt1romseg
```

When traced by a logic analyzer, with 0000h as the address of 'control', these are the bus cycles the 80386 generates:

```
 Line  addr,H   80386 Mnemonic
-----  -------- ------------------------
   4   0000103c 00a1c08e code read
   5   00001040 00003d00 code read
   6   00001044 013df874 code read
   7   00001048 90187500 code read
   8   00000000 90180000 memory read
   9   0000104c 16000f90 code read
  10   0000103c 00a1c08e code read
  11   00001040 00003d00 code read
```

The above trace list shows several features of the 80386 bus activity:

- Even though readloop begins at address 103e, the processor had to fetch instructions starting at address 103c each time it jumped to readloop. The 80386 always reads four bytes when reading instructions.
- The processor prefetched 12 bytes of instructions (addresses 1040 through 1048) before executing the 'mov ax,control' instruction at address 103e. You can see this by seeing that the read of 'control' (address 0) occurs at state 8, not after state 5 where the entire opcode had been read.
- Even after 'control' was read, the processor continued to prefetch, reading address 104c at state 9 in the trace before recognizing it had to jump back to address 103e.

Disassembly helps

Fortunately, the disassembler which is part of RTC helps you decode the order of execution. Here is the output of the 'trace' command, displaying disassembled bus cycles:

```
 4    0000103c 00a1c08e -MOV ES,AX
     =0000103e          MOV AX,0000H
 5    00001041 00003d00 CMP AX,#0000H
 6    00001044 013df874 JZ 0000103EH
     =00001046          -CMP AX,#0001H
 7    00001048 90187500 -  90187500H  code read
 8    00000000 xxxx0000 xxxx0000H  read mem
 9    0000104c 16000f90 -  16000F90H  code read
10    0000103c 00a1c08e -MOV ES,AX
     =0000103e          MOV AX,0000H
11    00001041 00003d00 CMP AX,#0000H
```

- The lines preceded by equals signs (=) did not appear as bus cycles. Instead, they were emitted by the disassembler. They were obtained as part of the preceding fetch.
- When a dash (-) is shown preceding a mnemonic, it indicates that the associated opcode was not executed. Instead, it was obtained in an unexecuted prefetch.
- When a multiple-byte opcode is decoded, the next address in the address column shows the starting byte of the next opcode, not the address that appeared on the address bus. This is convenient when using an assembly listing to match up addresses, but you cannot trigger a trace on this address. Only use addresses that are multiples of four when specifying a trigger for the analyzer.

Execution Trace Messages help even more

In many cases, the disassembler cannot correctly determine which bytes are unused prefetches and which are executed. The "execution trace message" facility in this emulator helps you make the determination.

When the "Enable Execution Trace Messages" box in the Settings→ Emulator Config→Hardware... dialog box is checked, the processor emits the target of any branches to the analyzer (use of "Enable Execution Trace Messages" has little or no effect on the performance of your target system.)

Consider the following code which jumps into a table based on the value in the ax register:

```
0140 40      53          inc ax
0141 BA4801  54          mov dx,offset table_start
0144 01C2    55          add dx,ax
0146 FFE2    56          jmp dx
0148 40      57  table_start:  inc ax
0149 42      58  entry2: inc dx
014A 41      59  entry3: inc cx
             60
014B EBFE    61          jmp $
```

These are the bus cycles when the above code is executed:

```
 Line  addr,H   80386 Mnemonic
-----  -------- ------------------------
   0   00000140    0148ba40H  code read
   1   00000144    e2ffc201H  code read
   2   00000148    eb414240H  code read
   3   00000148    eb414240H  code read
   4   0000014c    909090feH  code read
   5   00000150    90909090H  code read
   6   00000154    90909090H  code read
   7   00000148    eb414240H  code read
   8   0000014c    909090feH  code read
```

The RTC disassembler helps, but it cannot identify the exact destination of the indirect jump, which could be the opcode at address 148, 149, 14a, or even 14b (because they were all fetched together). There is no way to tell without knowing the value of register AX at the start of the trace, and there is no hint as to its starting value.

```
0  00000140 0148ba40 INC AX
  =00000141          MOV DX,#0148H
1  00000144 e2ffc201 ADD DX,AX
  =00000146          JMP NEAR PTR DX
2  00000148 eb414240 -  EB414240H  code read
3  00000148 eb414240 INC AX
  =00000149          INC DX
  =0000014a          INC CX
  =0000014b          JMP  0000014BH
4  0000014c 909090fe -  909090FEH  code read
5  00000150 90909090 -  90909090H  code read
6  00000154 90909090 -  90909090H  code read
7  00000148 eb414240 -INC AX
  =00000149          -INC DX
  =0000014a          -INC CX
  =0000014b          JMP  0000014BH
```

Note that the disassembler had no choice but to assume that the jump was to address 148 in the above trace (the first byte fetched). By enabling Execution Trace Messages, the disassembler can produce the following trace list:

```
0  00000140 0148ba40 INC AX
  =00000141          MOV DX,#0148H
1  00000144 e2ffc201 ADD DX,AX
  =00000146          JMP NEAR PTR DX
2  00000148 eb414240 -  EB414240H  code read
3  00000148 eb414240 -INC AX
  =00000149          INC DX
  =0000014a          INC CX
  =0000014b          JMP  0000014BH
4  00000149 xxxx42xx branch trace msg, dest=00000149H
```

In this listing, the "branch trace message" shows that the indirect jump went to address 149. Note that the instruction at address 148 is an unused prefetch, accurately marked by the leading dash. (By reading this trace list, you can see that register AX must have been equal to '0' on entry to this routine.)

Using Execution Trace Messages to observe program flow

If you enable Execution Trace Messages, and then store only cycles with the 'btmsg' status, you can obtain a concise trace showing the exact "flow" of your program. Only branches that are taken will appear, so you can observe calls to your functions, returns from them, "if" statements executed, and the number of times loops are executed. Since only the branches are stored, you can keep a record of program activity for a very long time before filling all of your analysis memory.

Understanding Address, Data, and Status

The 80386 has a 32-bit data bus but allows the program to access data contents in 8-bit, 16-bit, and 32-bit segments. It can be difficult to know how to define a specification for the external bus on the 80386 when you want to perform a trace. The following information will help you decide what to put in the A:D:S: fields of the analyzer in order to trigger, store, or sequence the analyzer to capture desired information.

Code fetches

If your hardware asserts BS16# low, the processor will do two fetches: one from address 4000, and the next from address 4002. This makes it difficult to specify an address for instruction fetches. In fact, bit 1 of any instruction address must be "don't care". This must be specified in binary. Otherwise, all four lower bits will be don't cares.

There are two cases where the emulator has been designed to know you want to "don't care" bit 1:

- When you specify an address and use the status "memif", the analyzer will "don't care" address bit 1, and set address bit 0 to 0.
- When you specify an address with a symbol, and that symbol is in a code segment, the address will be "don't cared" correctly. If you do not wish this to happen, use "<symbol>+0".

Example:

If address 5 contains an op-code, the analyzer must trigger on 01x0 binary. If this was entered as "A: 5 S: memif", the correct trigger specification will be entered automatically. If address 5 was the symbol "START", simply using the symbol will also automatically generate the address pattern 01x0 instead of 0101. If this was entered as "START+0", the trigger address pattern will be 0101.

Data read/write

Data values are 32-bit values (because the data bus is 32 bits wide). To identify byte values on the data bus, use "don't cares" (x) as shown below:

- Data at multiple of 4 (e.g. 0, 4, 8): 0xxxxxx12
- Data at multiple of 4 + 1 (e.g. 1, 5, 9): 0xxxx34xx
- Data at multiple of 4 + 2 (e.g. 2, 6, A): 0xx56xxxx
- Data at multiple of 4 + 3 (e.g. 3, 7, B): 078xxxxxx

For example, to specify a write to address bye 4032 with value 23:

```
Address:4032 Data:0xx23xxxx Status:write
```

Take similar care for 16-bit data:

- Data at multiple of 4 (e.g. 0, 4, 8): 0xxxx3412
- Data at multiple of 4 + 1 (e.g. 1, 5, 9): 0xx5634xx
- Data at multiple of 4 + 2 (e.g. 2, 6, A): 07856xxxx
- Data at multiple of 4 + 3 (e.g. 3, 7, B) is a special problem.

The 80386 will have to generate multiple bus cycles to do the access for data at multiple of 4 + 3. Depending on your needs, one of the following four choices should be right for you:

- Trace only the first access: 078xxxxxx.
- Trace only the second access: 0xxxxxx9a.
- Trace both accesses in any sequence: 078xxxxxx or 0xxxxxx9a.
- Trigger only if a specific access is followed by the other. See Trace→Sequence... (ALT, T, Q) for more information.

If 32-bit data is not 32-bit aligned, you will see problems similar to those of the 16-bit data at a multiple of 4 + 3.

Status values identify the types of 80386 bus cycles. Status values may be ANDed together by selecting two or more in the Trace Pattern Dialog Box, accessable via the Trace→Edit... dialog box and the Trace→Sequence... dialog box. For example, to trigger on the occurrence of a data read immediately following a HOLD cycle, select hlda, memrd, and data together.

If you need a combination of status values not available in the predefined list, you may compose a binary value from the following information:

Status Bits 31 - 16	Description
xxxx xxxx xxxx xxx0	0 = monitor cycle. See Note 1 below.
xxxx xxxx xxxx xx1x	1 = write. 0 = read.
xxxx xxxx xxxx x1xx	1 = data. 0 = code.
xxxx xxxx xxxx 1xxx	1 = memory. 0 = data.
xxxx xxxx xxx0 xxxx	0 = Byte Enable 0 (bits 7..0) active.
xxxx xxxx xx0x xxxx	0 = Byte Enable 1 (bits 15..8) active.
xxxx xxxx x0xx xxxx	0 = Byte Enable 2 (bits 23..16) active.
xxxx xxxx 0xxx xxxx	0 = Byte Enable 3 (bits 31..24) active.
xxxx xxx0 xxxx xxxx	0 = Bus Size 16-pin active.
xxxx xx0x xxxx xxxx	0 = NA pin (pipelining) active.
xxxx x0xx xxxx xxxx	0 = LOCK pin active.
xxxx 1xxx xxxx xxxx	1 = PEREQ pin active.
xxx0 xxxx xxxx xxxx	0 = BUSY pin active.
xx0x xxxx xxxx xxxx	0 = ERROR pin active.
x1xx xxxx xxxx xxxx	1 = INTR pin active.
1xxx xxxx xxxx xxxx	1 = HLDA was active before this cycle.

Note 1: Also controls cycle type in the Settings→Extended→Trace menu.

Entering Addresses as Constants

This chapter contains information about entering addresses as constants (instead of using symbols or clicking on source lines in the source display)

- Overview of 80386 address types
- Explanation: why different syntax for different address types
- Syntax guide for constant-addresses

Overview of 80386 address types

The 80386 uses several different types of addresses. This section gives a brief definition of each type. For more information, see your 80386 programmer's handbook.

Physical addresses

These are the addresses actually available on the address pins of the 80386. They are used by the memory and I/O subsystems on an 80386-based system. They have 32 bits on the 80386DX, and 24 bits on the 80386CX and 80386EX.

Linear addresses

These are the addresses used by the hardware breakpoints on the 80386, and are inputs into the paging hardware on the 80386. They have 32 bits.

Virtual addresses

These are the addresses as seen by the programmer. There are three types of virtual addresses: real-mode, protected-mode, and virtual 8086-mode.

Real-mode These addresses have a 16-bit segment and a 16-bit offset. The linear address is calculated as: (segment * 64) + offset. After RESET, the processor is in 'real mode'. In this mode, physical addresses are the same as linear addresses.

Protected-mode These addresses have a 32-bit selector and a 16-bit or 32-bit offset. The linear address is calculated by using 13 bits of the selector as an index into the GDT (Global Descriptor Table), reading a base address from that entry in the GDT, and adding the offset.

Virtual 8086-mode These addresses have a 16-bit segment and a 16-bit offset. The linear address is calculated as: (segment * 64) + offset. In this mode, paging can be used, so the physical address is not necessarily the same as the linear address.

Explanation: why different syntax for different address types

There are several reasons why this emulator differentiates between real-mode addresses and protected-mode addresses:

- To reduce the use of the monitor when doing dynamic translations. Real-mode addresses do not need to traverse any tables, but protected-mode addresses do.
- To allow the use of protected-mode addresses while the processor is still in real mode (e.g. it is reset). This is generally used to set up breakpoints or to set up a trace.

To allow clear display of real-mode addresses versus protected-mode addresses.

Constant-address syntax

Physical addresses

offset — offset is a 32-bit value.

Real-mode addresses

segment:offset — Segment is a 16-bit value, and offset is a 16-bit value. The linear address is calculated as: (64 * segment) + offset

Protected-mode addresses, GDT only

selector::offset — Offset is 16 or 32 bits; selector is an entry into the GDT (current or cached)

Protected-mode addresses, GDT and LDT specified

selector:ldt:offset — Offset is 16 or 32 bits; selector is an entry into the GDT (current or cached) which points to an LDT; ldt is the entry in that LDT.

All 16 and 32-bit values are entered as numeric constants.

See Also

"Selecting how Address Translations work" in the "Configuring the Emulator" chapter.

Unexpected Stepping Behavior

The emulator uses the single-step trap feature of the i83086 processor to single step instructions. A single-step trap happens when:

- The TF flag in the EFLAGS register is set.
- An instruction is executed with the TF flag set.

Faults

If an instruction causes a fault, the flags register is saved on the stack and the TF flag is cleared *before* the fault handler is executed. Unless the fault handler restores the value of the TF flag saved on the stack, the entire fault handler will be executed without generating a single-step trap.

For example, if a floating-point instruction is executed on a system that does not have an i80387 floating-point coprocessor, an instruction fault will be generated. This type of fault is typically fielded by a floating-point emulation library which processes the exception stack frame, decodes and emulates the floating-point instruction, modifies the return address on the stack to point to the next instruction, and returns from the fault handler. However, because no instructions were executed with the trap flag set, the processor does not generate a single-step trap. The processor will continue to execute floating-point instructions until the first normal instruction is executed.

This does not occur when floating-point instructions are executed on an i80387 coprocessor. Floating-point emulation libraries could be implemented to generate a single-step trap upon return by restoring the TF flag from the stack immediately prior to executing the IRET/IRETD instruction.

INT instructions

Like an instruction fault, the TF flag is saved on the stack and then cleared prior to execution of the first instruction in the interrupt handler. Therefore, on returning from the INT instruction, the processor will execute the next instruction, and then generate the single-step fault (assuming the next instruction is not another INT, fault, etc.).

Task gates

If the instruction is a task gate, the EFLAGS register is saved in the old TSS and the TF flag is restored from the new TSS prior to execution of the first instruction of the new task. Therefore, the entire task will be executed before the single-step trap occurs.

To step into a task or a fault handler

First, set a breakpoint in the routine you want to step into. Then do a "run" command. If you do a step as you go into the INT routine or the fault handler, the TF flag will be restored when you return from the INT routine or fault handler routine. This means that if you do a RUN while in the routine, you will enter the monitor on the instruction after the routine returns.

Part 5

Installation Guide

Instructions for installing the product.

14

Installing the Debugger

Installing the Debugger

This chapter shows you how to install the Real-Time C Debugger.

- Requirements
- Before Installing the Debugger
- Step 1. Connect the HP 64700 to the PC
- Step 2. Install the debugger software
- Step 3. Start the debugger
- Step 4. Check the HP 64700 system firmware version
- Optimizing PC Performance for the Debugger

Requirements

- IBM compatible or NEC PC with an 80486 class microprocessor and 8 megabytes of memory.
- MS Windows 3.1, set up with 20 megabytes of swap space.
- VGA Display.
- 3 Megabytes available disk space.
- Serial port, HP 64037 RS-422 port, or Novell LAN with Lan Workplace for DOS or Microsoft Lan Manager with HP ARPA Services.
- Revision A.04.00 or greater of HP 64700 system firmware. The last step in this chapter shows you how to check the firmware version number.

Before Installing the Debugger

- Install MS Windows according to its installation manual. The Real-Time C Debugger must run under MS Windows in the 386 enhanced mode.

- 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.

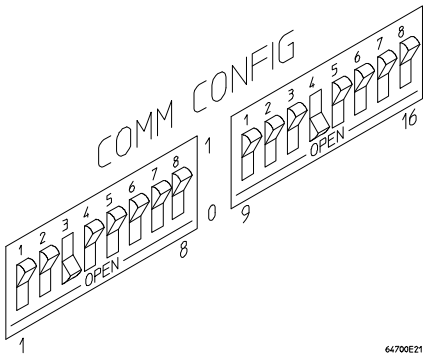
To connect via RS-232

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.

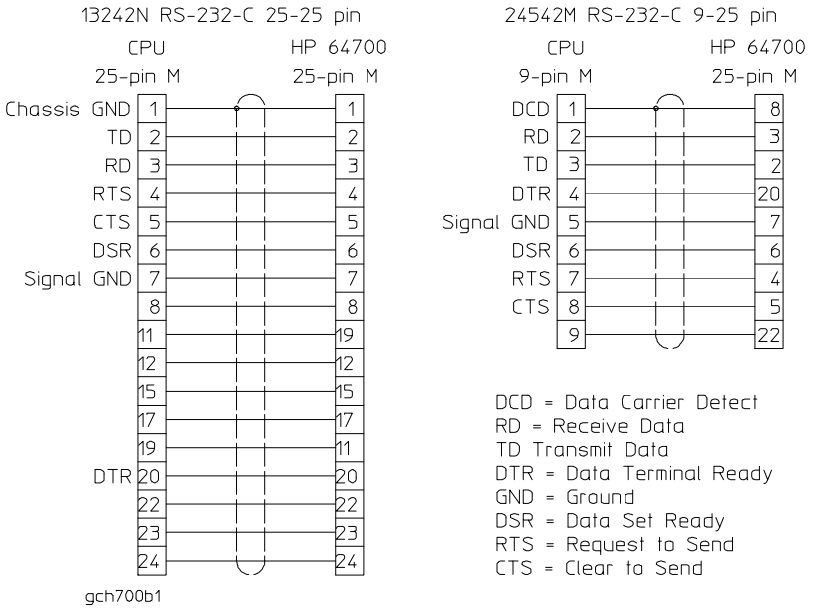


Notice that switches 1 through 3 are set to 001, respectively. This sets the baud rate to 19200.

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.



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: 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 Enter 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.

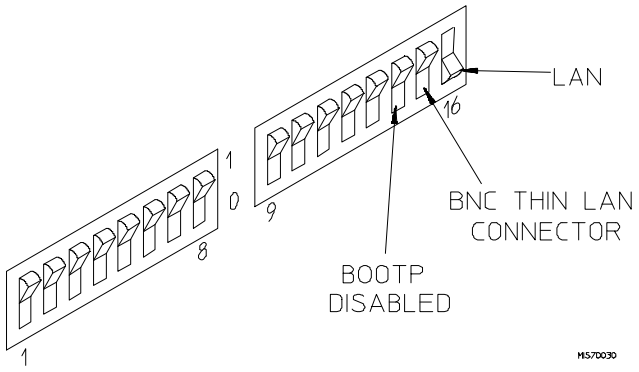
2 Exit the Terminal or telnet program.

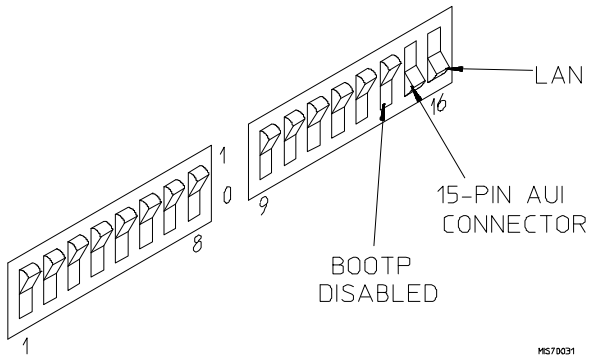
3 Turn OFF power to the HP 64700.

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.

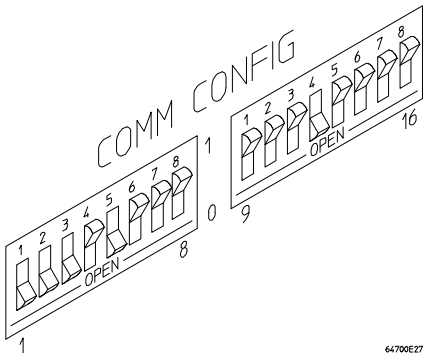
7 Verify LAN communication by using a "telnet <HP 64700 IP address>" command. This connection will give you access to the HP 64700 Terminal Interface.

You should now be able to press the Enter key in the telnet window to see the HP 64700's Terminal Interface prompt (for example, R>, M>, U>, etc.). If you see the prompt, you have verified LAN communication. If you cannot connect to the HP 64700's IP address, refer to If you cannot verify LAN communication.

To connect via RS-422

Before you can connect the HP 64700 to the PC via RS-422, the HP 64037 RS-422 Interface must have already been installed into the PC.

1 Set the HP 64700 configuration switches for RS-422 communication. Locate the DIP switches on the HP 64700 rear panel, and set them as shown below.



Notice that switches 1 through 3 are set to 111, respectively. This sets the baud rate to 230400.

Notice that switch 5 is set to 1. This configures the 25-pin port for RS-422 communication.

Notice also that switches 12 and 13 are set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake which is needed to make sure all characters are processed.

2 Connect the 17355M cable (which comes with the HP 64037 interface) from the PC to the HP 64700.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel. The power 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 or 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 or 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.

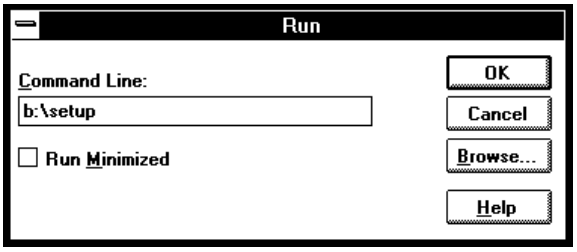
Step 2. Install the debugger software

1 If you are updating or re-installing the debugger software, you may want to save your B3637.INI file because it will be overwritten by the installation process.

2 Start MS Windows in the 386 enhanced mode.

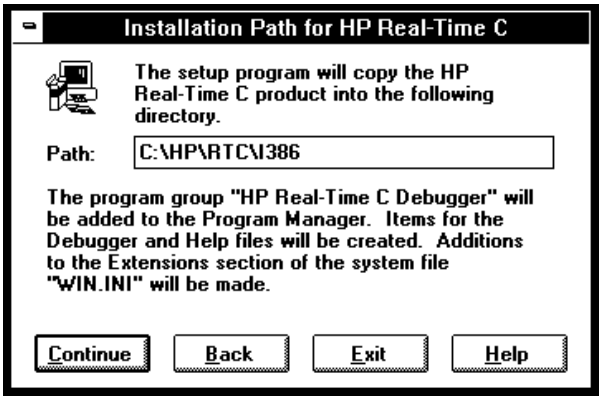
3 Insert the 80386 REAL-TIME C DEBUGGER Disk 1 of 2 into floppy disk drive A or B.

4 Choose the File→Run... (ALT, F, R) command in the Windows Program Manager. Enter "a:\setup" (or "b:\setup" if you installed the floppy disk into drive B) in the Command Line text box.

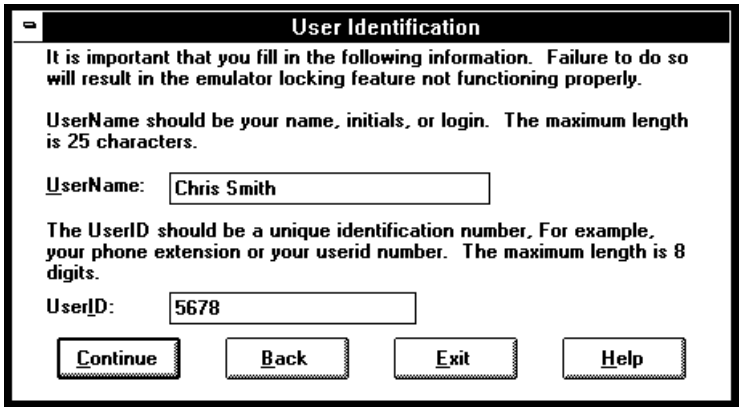


Then, choose the OK button. Follow the instructions on the screen.

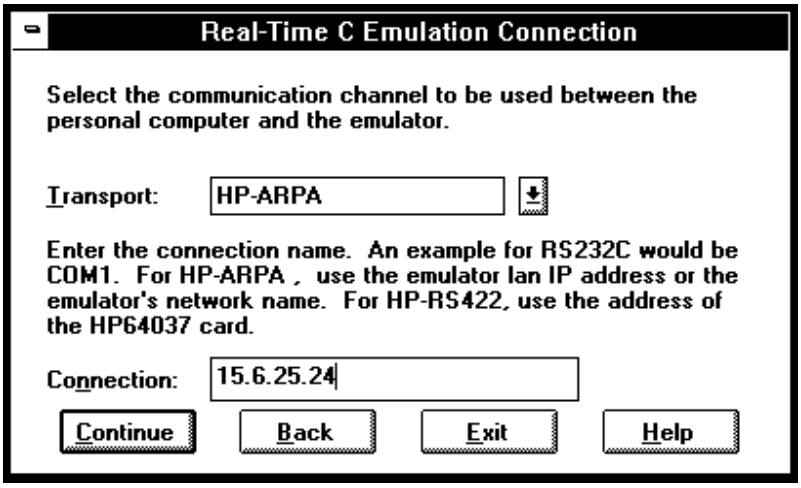
You will be asked to enter the installation path. The default installation path is C:\HP\RTC\I386. The default installation path is shown wherever files are discussed in this manual.



You will be asked to enter your user ID. This information is important if the HP 64700 is on the LAN and may be accessed by other users. It tells other users who is currently using, or who has locked, the HP 64700. This information can be modified while using the Real-Time C Debugger by choosing the Settings→Communication... (ALT, S, C) command.



You will be asked to select the type of connection to be made to the HP 64700. This information can be modified while using the Real-Time C Debugger by choosing the Settings→Communication... (ALT, S, C) command.



When using the HP-RS422 transport, the connection name is the I/O address you want to use for the HP 64037 card. Enter a hexadecimal number from 100H through 3F8H, ending in 0 or 8, that does not conflict with other cards in your PC.

After you have specified the type of connection, files will be copied to your hard disk. (The B3637.TMP and B3637.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 "I80386 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) to set up the "Communications..." settings as follows:

```
Baud Rate" 19200 (or whatever you have chosen for the
emulator)
Data Bits: 8
Parity: None
Flow Control: Hardware
Stop Bits: 1
```

When you are connected, press the Enter key. You should get a prompt back. If nothing echos back, check the switch settings on the back of the emulator:

Switches 1 thru 3 set the baud rate as follows:

```
S1   S2   S3
0    0    0    9600
0    0    1    19200
0    1    0    2400
```

Switches 12 and 13 must be set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake which is needed to make sure all characters are processed.

All other switches should be in the "0" position, especially switch 16 on the HP 64700 (which selects LAN/Serial interface).

Remember that if you change any of the switch positions, you must turn OFF power to the HP 64700 and turn it ON again before the changes will take effect.

☐ **If the switches are in the correct position and you still do not get a prompt when you press return, check the following:**

- Turn OFF power to the HP 64700 and then turn it ON again. Press return to see if you get a prompt.
- Check to make sure the RS-232 cable is connected to the correct port on your PC, and that the cable is appropriate for connecting the PC to a DCE device. If the cable is intended to connect the PC to a DTE device, set switch 4 to "1" (which makes the emulator a DTE device), turn OFF power to the HP 64700, turn power ON, and try again.
- Check to make sure your RS-232 cable has the RTS, CTS, DSR, DCD, and DTR pins supported. If your PC RS-232 connection is a 9-pin male connection, HP cable number 24542M will work (set switch 4 to 0 if you use this cable). If your PC has a 25-pin RS232 connector, HPO cable number13242N will work (set switch 4 to 0).

When using certain RS-232 cards, connecting to an RS-232 port where the HP 64700 is turned OFF (or not connected) will halt the PC. The only way to restore operation is to reboot the PC. Therefore, HP recommends you always turn ON the HP 64700 before attempting to connect via RS-232.

If 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 cycle power on 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 inportant 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 mast 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 mast>' command.

"lan -p"; shows the port is 6470. If the port is not 6470, you must change it with the 'lan -p 6470' command.

Both the PC's subnet mask and the emulator's subnet mask must be identical unless they communicate via gateway or a bridge. Unless your Network

Administrator states otherwise, make them the same. If you are using HP-ARPA, you can check the PC's subnet mask with the 'lminst' command. If you are using Novell LAN WorkPlace, make sure the file \NET.CFG has the entry "ip_netmask <subnet mask>" in the section "Protocol TCPIP." If you are using Windows for Workgroups, you can check the PC's subnet mask by looking in the [TCPIP] section of the PROTOCOL.INI file or by looking in the Microsoft TCP/IP Configuration dialog box. If you are using WINSOCK, refer to your LAN software documentation for subnet mask information.

☐ **Occasionally the emulator or the PC will "lock up" the LAN due to excessive network traffic. If this happens, all you can do is turn OFF power to the HP 64700 or PC, 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 LAN DLL errors

The various LAN transport selections require the following DLLs:

HP-ARPA	WSOCKETS.DLL.
Novell-WP	WLIBSOCK.DLL.
W4WNG-TCP	WSOCKETS.DLL. (Windows for Workgroups)
WINSOCK1.1	WINSOCK.DLL.

These DLLs are included with LAN software. The required DLL must be in your search path. This will be the case if your network software is installed.

If you have RS-422 connection problems

☐ Make sure the HP 64700 switch settings match the baud rate chosen when attempting the connection.

Switches 1 thru 3 set the baud rate as follows:

S1	S2	S3	
1	1	1	230400
1	1	0	115200
1	0	1	38400
1	0	0	57600
0	1	1	1200
0	1	0	2400
0	0	1	19200
0	0	0	9600

Switch 5 must be set to 1 to configure the HP 64700 for RS-422 communication.

Switches 12 and 13 must be set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake which is needed to make sure all characters are processed.

All other switches should be in the "0" position, expecially 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 turning it ON again.

☐ If you still don't get a prompt, make sure the HP 17355M RS-422 cable is connected to the correct port on your PC.

Step 4. Check the HP 64700 system firmware version

- Choose the Help→About Debugger/Emulator... (ALT, H, D) command.

  The version information under HP 64700 Series Emulation System must show A.04.00 or greater. If the version number is less than A.04.00, you must update your HP 64700 system firmware as described in the Installing/Updating HP 64700 Firmware chapter.

Optimizing PC Performance for the Debugger

The Real-Time C Debugger is a memory and I/O intensive Windows program. Slow user interface performance may be caused by many things:

- Underpowered PC -- The Real-Time C Debugger requires an IBM compatible or NEC PC with an 80486 class microprocessor, 8 megabytes of memory, and 20 megabytes of MS Windows swap space. Because RAM is faster than swap, performance is best when there is enough RAM to accommodate all of the Real-Time C Debugger's memory usage (which is directly related to the size of your programs and the amount of debug information in them).
- Improperly configured PC -- Windows configuration may have a very significant effect on performance. The Windows swap file settings are very important (see the Virtual Memory dialog box under 386 Enhanced in the Control Panel). The larger the swap file, the better the performance. Permanent swap has superior performance.
- Disk performance (due to Windows swap file access and Windows dialog and string resource accesses from the debugger ".EXE" file) -- The disk speed has a direct impact on performance of the Real-Time C Debugger. Use of SMARTDrive or other RAM disk or caching software will improve the performance.

Various PC performance measurement and tuning tools are commercially available. Optimizing your PC performance will improve debugger interface performance and, of course, all your other PC applications will benefit as well.

15

Installing/Updating HP 64700 Firmware

Installing/Updating HP 64700 Firmware

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

Note

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

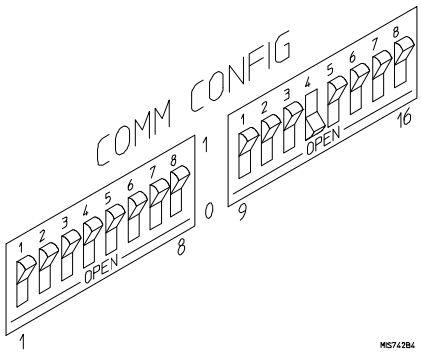
The firmware, and the program that downloads it into the HP 64700, are included with the debugger on floppy disks labeled HP 64700 EMULATION AND ANALYSIS FIRMWARE.

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.



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 Enter 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.

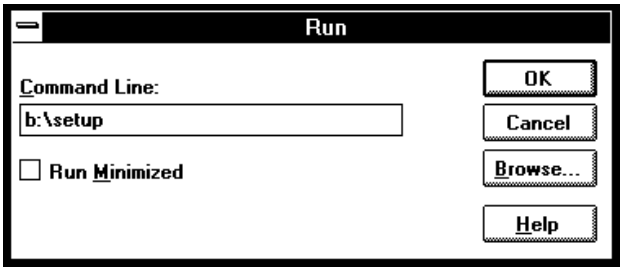
Step 2. Install the firmware update utility

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.

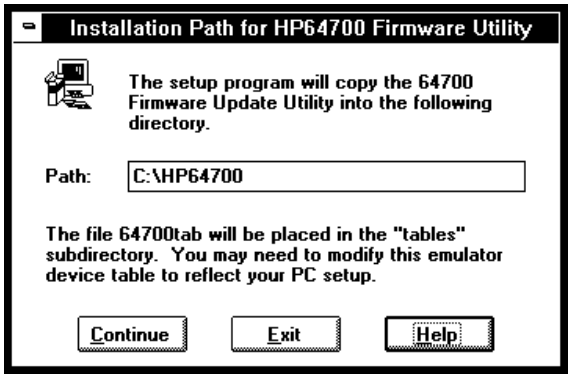
2 Insert the HP64700 EMUL/ANLY FIRMWARE Disk 1 of 2 into floppy disk drive A or B.

3 Choose the File→Run... (ALT, F, R) command in the Windows Program Manager. Enter "a:\setup" (or "b:\setup" if you installed the floppy disk into drive B) in the Command Line text box.



Then, choose the OK button. Follow the instructions on the screen.

You will be asked to enter the installation path. The default installation path is C:\HP64700.



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
```

7 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. Use of the background monitor does not consume any processor resources; the monitor is in a separate address space.

breakaddress A breakaddress is an address where a breakpoint has been set. It may be an address, a line number, or a line_number.macro_number (example 34.1).

break on trigger Causes emulator execution to break into the monitor when the trigger condition is found. This is known as a hardware breakpoint, and it lets you break on a wider variety of conditions than a software breakpoint (which replaces an opcode with a break instruction); however, depending on the speed of the processor, the actual break point may be several cycles after the one that caused the trigger.

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

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

control menu The menu that is accessed by clicking the control menu box in the upper left corner of a window. You can also access control menus by pressing the "ALT" and "-" keys.

count condition Specifies whether time or the occurrences of a particular state are counted for each state in the trace buffer.

Device Register Window This window does not appear in the Real-Time C Debugger for the 80386.

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 (target) programs execute. Foreground memory is made up of dual-ported memory that resides in the emulator. Therefore, it does not use any target-system memory, but it does use target-system memory address ranges.

foreground monitor program An emulation monitor program that executes out of the same memory system as user (target) 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 (not background monitor programs) in emulation memory.

guarded memory Memory locations that should not be accessed by user programs. These locations are specified when mapping memory. If the user program accesses a location mapped as guarded memory, emulator execution breaks into the monitor.

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

monitor A program, executed by the emulation microprocessor (as directed by the emulation system controller), that gives the emulator access to target system memory, microprocessor registers, and other target system resources.

object file An Intel OMF386 boot-loadable format absolute file that can be loaded into emulation or target system memory and executed by the debugger.

pop-up menu A menu that is accessed by clicking the right mouse button in a window.

prestore condition Specifies the states that may be stored before each normally stored state. Up to two states may be prestored for each normally stored state.

primary branch condition Specifies a condition that causes the analyzer to begin searching at another level.

restart condition Specifies the condition that restarts the two-step sequential trigger. In other words, if the restart condition occurs while the analyzer is searching for the trigger condition, the analyzer starts looking for the enable condition again.

secondary branch condition Specifies a condition that causes the analyzer to begin searching at another level. If a state satisfies both the primary and secondary branch conditions, the primary branch will be taken.

sequence levels Levels in the analyzer that let you specify a complex sequential trigger condition. For each level, the analyzer searches for primary and secondary branch conditions. You can specify a different store condition for each level. The Page button toggles the display between sequence levels 1 through 4 and sequence levels 5 through 8.

state qualifier A combination of address, data, and status values that identifies particular states captured by the analyzer.

status values Values that identify the types of microprocessor bus cycles recognized by the analyzer. You can include status values (along with address and data values) when specifying trigger and store conditions. The status values defined for the HP i80386 emulator are:

be0	BE0# (Byte Enable 0) active
be1	BE1# (Byte Enable 1) active
be2	BE2# (Byte Enable 2) active
be3	BE3# (Byte Enable 3) active
bs16	BS16# (Bus Size 16) active
btmsg	Branch Trace Message
busy	BUSY# active (from the coprocessor)
ctrl	A control access (op-code fetch, for example)
data	A data access (memory read, for example)
error	ERROR# active (from the coprocessor)
halt	The 'hlt' instruction was executed
hlda	HLDA (hold acknowledge) was active just prior to captured state (a DMA occurred)
inta	An interrupt acknowledge cycle

intr	INTR (interrupt request) line is active
io	An I/O access ('out', for example)
iord	An I/O read cycle
iowr	An I/O write cycle
lock	LOCK# (locked cycle)
mem	A memory access ('read', for example)
memif	A memory instruction fetch (op-code fetch)
memrd	A memory read
memwr	A memory write
mon	A background monitor cycle
na	NA# (pipelining) request active
pereq	PEREQ active (from the coprocessor)
read	A read cycle (memory or I/O)
shut	Processor shutdown
ttmsg	Task Trace message
write	A write cycle (memory or I/O)

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 appears at the start, center, or end of the trace buffer. In other words, the trigger position specifies whether states are stored after, about, or before the trigger.

trigger store condition Specifies which states get stored in the trace buffer while the analyzer searches for the trigger condition.

watchpoint A variable that has been placed in the WatchPoint window where its contents can be readily displayed and modified.

Index

A
abort, during object file or memory load, 268
accumulated count information, displaying, 125, 323
Add to Watch command, 337
address
  specification for tracing, 395-397
  syntax: why different for different types, 400
  translations, 254-257
  translations, mapping for demo program, 8
addresses
  searching, 77, 309
  when they are translated, 62-63
A:D:S:, 395-397
analysis of 80386 bus cycles, 391-394
analyzer, 439-444
  editing the trace specification, 139, 212
  halting, 123, 225
  repeating last trace, 123, 226
  setting up with "Find Then Trigger", 130, 216-219
  setting up with "Sequence", 134, 220-223
  setting up with "Trigger Store", 127, 213-215
  setup, 395-397
  trace signals, 389-390
  tracing until halt, 123, 224
arguments, function, 364
arm condition, 64, 130, 134, 227-229, 262, 439-444
arrays (C operators), 159
ASCII values in Memory window, 102, 369
Assemble... (ALT, A) command, 243
assembler, in-line, 243
assembly language instructions
  stepping multiple, 89, 188-190
stepping single, 87, 186
auto variables, 99-101
AUTOEXEC.BAT file, 435-436

B
background memory, 439-444
background monitor, 386-388
  program, 439-444
background monitor, selecting, 54, 251-253
background operation, tracing, 266-267
BackTrace window, 364
  displaying source files, 335
baud rate
  RS-232, 350
  RS-422, 350
beep, sounding from command file, 341
BNC port
  driving the trigger signal, 261
  output trigger signal, 64
  receiving an arm condition from, 262
  receiving an arm condition input, 64
  setting up, 64
BP marker, 11, 13, 96, 193-198, 376
branch trace messages, 391-394
  enabling, 244-246
break into monitor, 91, 191
break macros, 439-444
  command summary, 146-151
  deleting, 98, 198
  listing, 96, 199-202
  setting, 96, 196-197
break on writes to ROM, enabling or disabling, 51
Breakpoint→Delete at Cursor (ALT, B, D) command, 195
Breakpoint→Delete Macro (ALT, B, L) command, 198
Breakpoint→Edit... (ALT, B, E) command, 199-202
Breakpoint→Set at Cursor (ALT, B, S) command, 193-194
Breakpoint→Set Macro... (ALT, B, M) command, 196-197

breakpoints, 439-444
deleting, 13, 95, 195
disabling, 95
enable or disable, 50, 244-246
hardware, 199-202
listing, 96, 199-202
setting, 11, 94, 193-194
software, 199-202
variable access, 199-202
variable modify, 199-202
bus cycles, 391-394
displaying, 124, 322
Button window, 365
editing, 38, 279
buttons that execute command files, creating, 38

C
C operators, 159
cable length, RS-232, 350
callers (of a function), tracing, 22, 119, 206-207
chain command files, 343
Clear Breakpoint command, 336
clipboard, 28
CLK2 speeds greater than 60 MHz, 48
clock speeds greater than 60 MHz, 48
command files
chain, 343
command summary, 46-151
comments, 345
creating, 36, 168
executing, 37, 171-172
executing at startup, 30, 37
exiting execution, 342
inserting wait delays, 347
locating cursor, 309
nesting, 343
parameters, 171-172
rerun, 344
sounding beep, 341
turning logging on or off, 169-170
command line options, 30, 32, 37
command summary, 146-151
commands, menu bar, 162

comments in command files, 345
communications (emulator), setting up, 258-260
CONFIG.SYS file, 435-436
configurations
  emulator, 244-246
  saving and loading, 65-66
Constant-Address Syntax, 398-401
Continuous Update (ALT, -, U) command, 300
control menu, 439-444
Copy→Destination... (ALT, -, P, D) command, 278
Copy→Registers (ALT, -, P, R) command, 300
Copy→Window (ALT, -, P, W) command, 277
count conditions, 227-229, 439-444
count information
  displaying accumulated, 125, 323
  displaying relative, 125, 323
CTRL key and double-clicks, 28
cursor, locating cursor from command file, 309
cut and paste, 28

D
data specification for tracing, 395-397
DCE or DTE selection and RS-232 cable, 419
debugger
  arranging icons in window, 270
  cascaded windows, 270
  exiting, 24, 31, 178
  exiting locked, 179
  installing software, 421-423
  opening windows, 271-272
  starting, 5, 30, 424-428
  startup options, 32
  tiled windows, 270
  windows, opening, 33
demo programs, 4
  loading, 9
  mapping memory, 7-8
  running, 12
  setting address translations, 8
DeMorgan's law, 227-229
dialog boxes, file selection, 180

directories
search path, 310
source, 274
disassembler, 391-394
display mode
mixed, 74
source only, 74
toggling, 303-304
Display→From State... (ALT -, D, F), 324
Display→Select Source... (ALT, -, D, L) command, 305
displaying state from specific byte within a state, 324
displaying trace from specified state, 324
Domain Name Resolution (DNR), 352
double-clicks and the CTRL key, 28
dual-port emulation memory, 439-444
dynamic variables, 203-204, 329, 385

E
embedded microprocessor system, 439-444
emulation memory, 439-444
copying target system memory into, 106, 292
emulation microprocessor, resetting, 92, 192
emulation monitor, 439-444
programs, 386-388
emulator, 439-444
emulator configuration, 46-53, 244-246
emulator configuration
loading, 66, 175
saving, 65, 176
emulator hardware options, setting, 47
emulator limitations, external DMA support, 60
emulator probe
plugging into the target system, 42
unplugging from demo target system, 41
enable
breakpoints, 244-246
condition, 439-444
or disable software breakpoints, 50
store condition, 439-444
target interrupts, 244-246
Entering addresses as constants, 398-401
entries, searching GDT/LDT/IDT for, 297

environment variables, 76
HP64700, 435-436
HPTABLES, 435-436
PATH, 435-436
environment
loading, 173
saving, 174
ethernet address, 414
Evaluate It command, 337
execution trace messages, 391-394
enabling or disabling, 52
enabling, 244-246
execution unexpected during single stepping, 402-403
Execution→Break (F4), (ALT, E, B) command, 191
Execution→Reset (ALT, E, E) command, 192
Execution→Run (F5), (ALT, E, U) command, 181
Execution→Run to Caller (ALT, E, T) command, 183
Execution→Run to Cursor (ALT, E, C) command, 182
Execution→Run... (ALT, E, R) command, 184-185
Execution→Single Step (F2), (ALT, E, N) command, 186
Execution→Step Over (F3), (ALT, E, O) command, 187
Execution→Step... (ALT, E, S) command, 188-190
exiting command file execution, 342
Expression window, 366
clearing, 282
displaying expressions, 283
expressions, 154
displaying, 283
externals, displaying symbol information, 81, 312

F
fetches, instruction, 391-394
file selection dialog boxes, 180
File→Command Log→Log File Name... (ALT, F, C, N) command, 168
File→Command Log→Logging OFF (ALT, F, C, F) command, 170
File→Command Log→Logging ON (ALT, F, C, O) command, 169
File→Copy Destination... (ALT, F, P) command, 177
File→Exit (ALT, F, X) command, 178
File→Exit HW Locked (ALT, F, H) command, 179
File→Load Debug... (ALT, F, D) command, 173
File→Load Emulator Config... (ALT, F, E) command, 175

File→Load Object... (ALT, F, L) command, 166-167
File→Run Cmd File... (ALT, F, R) command, 171-172
File→Save Debug... (ALT, F, S) command, 174
File→Save Emulator Config... (ALT, F, V) command, 176
firmware
  update utility, installing, 435-436
  update, connecting the HP 64700 to the PC, 433-434
  using PROGFLASH to update, 437
  version information, 273
font settings, 263
  changing, 35
foreground memory, 439-444
foreground monitor, 386-388
  advantages and disadvantages, 386-388
  program, 439-444
  selecting, 55-56, 251-253
  traced as user, enabling or disabling, 53
foreground operation, tracing, 266-267
function arguments, 364
function keys, 29
functions
  displaying symbol information, 80, 312
  running until return, 18, 90, 183
  searching, 77, 307
  stepping over, 18, 88, 187
  tracing callers, 22, 119, 206-207
  tracing execution within, 121, 208-209

G gateway address, 414
GDT editing, 110
GDT to physical address translation, 62-63, 254-257
GDT window, 371
  displaying, 109
Getting Started, 4
global assembler symbols, displaying, 83, 314
Global Descriptor Table window, 371
global symbols, displaying, 81, 312
global variables, 81, 122, 312
glossary, 439-444
guarded memory, 60, 247-250, 379, 439-444

H hardware
breakpoints, 193-194, 199-202
locking on exit, 179
options, setting, 47
requirements, 409
Help→About Debugger/Emulator... (ALT, H, D) command, 273
hostname, 258-260
hosts file, 352-353
HP 64700
connecting to the PC, 411-420
connecting via LAN, 414
connecting via RS-232, 411
connecting via RS-422, 418
environment variable, 435-436
firmware update utility, installing, 435-436
firmware update, connecting the HP 64700 to the PC, 433-434
firmware, using PROGFLASH to update, 437
HP Probe, 352
HP-ARPA LAN address, 352
HPTABLES environment variable, 435-436

I I/O locations
displaying, 111
editing, 112
guarding, 237-238
specifying, 284
I/O window, 367
turning polling ON or OFF, 69
icons
for a different emulator, 32
(debugger window), arranging, 270
IDT editing, 110
IDT window, 374
displaying, 109
in-line assembler, 243
installation path, 421-423
instruction fetches, 391-394
internals, displaying symbol information, 82, 313
Internet Address, 258-260, 414, 420
Interrupt Descriptor Table window, 374

interrupts
enabling or disabling from target system, 49
(target system), 386-388
enabling, 244-246
interset operators, 227-229
intraset operators, 227-229
intrusion, monitor, 68, 235-236
inverse assembly with 16- or 32- bit segments assumed, 324

L labels, 156, 243
LAN
address format, 352-353
cards, 409-410
communication, 258-260, 424-428
connecting HP 64700, 414
LBG flying lead, connecting, 43
LDT editing, 110
LDT to physical address translation, 62-63
LDT to physical address translations, 254-257
LDT window, 373
displaying, 109
levels, trace sequence, 134, 139, 220-223, 234
limitations, Symbol window, 382
line (source file), running until, 19, 90, 182
linear address: definition, 399
link level address, 414
list file
changing the destination, 34
copying window contents to, 34
listing files, specifying, 177, 278
local assembler symbols, displaying, 83, 314
Local Descriptor Table window, 373
local symbols, displaying, 82, 313
local variables, 82-83, 313
lock hardware on exit, 179
log (command) files, 36, 168-172
logical operators, 130, 134, 227-229
logical to physical address translations, 254-257

M
macro, 439-444
memory
abort during load, 268
copying, 105, 290
copying target system into emulation memory, 106, 292
displaying, 102
editing, 104
loading from stored file, 294
mapper, resolution, 60
mapping, 58-61, 247-250
mapping for demo program, 7-8
modifying a range, 107, 291
searching for a value or string in, 108
storing to a binary file, 295
type, 60, 247-250
Memory window, 369
displaying 16-bit values, 287
displaying 32-bit values, 287
displaying bytes, 287
displaying multicolumn format, 287
displaying single-column format, 286
turning polling ON or OFF, 69
Menu Bar Commands, 162
microprocessor, resetting, 92, 192
mixed display mode, 74, 303
monitor, 439-444
enabling trace of foreground monitor, 53
intrusion, 68, 235-236
programs, 386-388
selecting the type, 54-57
multiple instructions execute in single step, 402-403

N
nesting command files, 343
no-operation command, 345
noabort, during object file or memory load, 268
Novell LAN address, 352
numeric constants, 155

O object file, 439-444
abort during load, 268
loading, 73, 166-167
loading the foreground monitor, 55-56
operators
C, 159
interset, 227-229
intraset, 227-229
logical, 130, 134, 227-229
options, command line, 32
output line
LBG, 43
RESET, 43
overview of 80386 address types, 399

P parameters, command file, 171-172
paste, cut and, 28
PATH environment variable, 435-436
path for source file
prompting, 269
search, 76, 310
patterns, trace, 130, 134, 216-223, 227-231
PC, connecting HP 64700, 411-420
performance (PC), optimizing for the debugger, 430
physical address: definition, 399
platform requirements, 409
pointers (C operators), 159
polling for debugger windows, turning ON or OFF, 69
pop-up menus, 439-444
accessing, 334-335
port, BNC, 64, 227-229, 261-262
port, communication, 258-260
power
turning OFF, 41
turning ON, 44
prefetching, 391-394
prestore condition, 130, 134, 216-223, 383, 439-444
primary branch condition, 134, 220-223, 439-444

Probe (emulator), 352
plugging into the target system, 42
unplugging from demo target system, 41
processor, resetting, 92, 192
PROGFLASH firmware update utility, 437
program counter, 87, 91, 181, 184-185, 188-190, 375-376
program modules, displaying symbol information, 80, 311
programs
demo, 4
loading, 73, 166-167
running, 91, 181, 184-185
stopping execution, 91

Q qualifier, state, 127, 213-215

R real-time mode
disabling, 68, 236
enabling, 68, 235
options, setting, 67-69
RealTime→I/O Polling→OFF (ALT, R, I, F) command, 238
RealTime→I/O Polling→ON (ALT, R, I, O) command, 237
RealTime→Memory Polling→OFF (ALT, R, M, F) command, 242
RealTime→Memory Polling→ON (ALT, R, M, O) command, 241
RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command, 236
RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D) command, 235
RealTime→Watchpoint Polling→OFF (ALT, R, W, F) command, 240
RealTime→Watchpoint Polling→ON (ALT, R, W, O) command, 239
Register window, copying information from, 300
registers
displaying, 20-21, 113
editing, 115
Register windows, 375
continuous update, 300
relative count information, displaying, 125, 323
requirements
hardware, 409
platform, 409
rerun command files, 344

reset
emulator, 92, 192
emulator status, 379
flying lead, connecting, 43
output line, 43
running from target system, 91, 184-185
slow to leave when using software breakpoints, 50
resolution, memory mapper, 60
restart condition, 130, 216-219, 439-444
return (function), running until, 18, 90, 183
ROM, enabling or disabling breaks on writes to, 51
RS-232
baud rate, 350
cable and DCE or DTE selection, 419
cable length, 350
communication error, 350
connecting HP 64700, 411
RS-422
baud rate, 350
connecting HP 64700, 418
run to caller, interrupting, 353
Run to Cursor command, 337

S
search path for source files, 76, 310
Search→Address... (ALT, -, R, A) command, 309
Search→Entry... (ALT, -, R, E) command, 297
Search→Function... (ALT, -, R, F) command, 307
Search→Selector... (ALT, -, R, S) command, 298
Search→String... (ALT, -, R, S) command, 306
Search... (ALT, -, R) command, 288
searching GDT/LDT/IDT
for entries, 297
for selector, 298
secondary branch condition, 134, 220-223, 439-444
selecting how address translations work, 62-63
selector, searching GDT/LDT/IDT for, 298
sequence levels, 234, 439-444
service ports, TCP, 414
Set Breakpoint command, 336

Settings→BNC→Input to Analyzer Arm (ALT, S, B, I) command, 262
Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, O) command, 261
Settings→Communication... (ALT, S, C) command, 258-260
Settings→Emulator Config→Address Translations... (ALT, S, E, A, 254-257
Settings→Emulator Config→Hardware... (ALT, S, E, H) command, 244-246
Settings→Emulator Config→Memory Map... (ALT, S, E, M) command, 247-250
Settings→Emulator Config→Monitor... (ALT, S, E, O) command, 251-253
Settings→Extended Settings→Load Error Abort→OFF (ALT, S, X, L, F) command, 268
Settings→Extended Settings→Load Error Abort→ON (ALT, S, X, L, O) command, 268
Settings→Extended Settings→Source Path Query→OFF (ALT, S, X, S, F) command, 269
Settings→Extended Settings→Source Path Query→ON (ALT, S, X, S, O) command, 269
Settings→Extended Settings→Trace Cycles→Both (ALT, S, X, T, B) command, 267
Settings→Extended Settings→Trace Cycles→Monitor (ALT, S, X, T, M) command, 266
Settings→Extended Settings→Trace Cycles→User (ALT, S, X, T, U) command, 266
Settings→Font... (ALT, S, F) command, 263
Settings→Symbols→Case Sensitive→OFF (ALT, S, S, C, F) command, 265
Settings→Symbols→Case Sensitive→ON (ALT, S, S, C, O) command, 265
Settings→Tabstops... (ALT, S, T) command, 264
side effects of software breakpoints, 50
single-step one line, 13
single-stepping, unexpected behavior, 402-403
software breakpoints, 193-194, 199-202
software breakpoints, enable or disable, 50
software, installing debugger, 421-423
Source at Stack Level command, 335
source directory, 274
source display mode, toggling, 303-304

source files
displaying, 10, 75, 305
displaying from BackTrace window, 335
prompting for paths, 269
searching for addresses, 77, 309
searching for function names, 77, 307
searching for strings, 78, 306
specifying search directories, 76
source lines
running until, 19, 90, 182
stepping multiple, 89, 188-190
stepping single, 87, 186
source only
displaying, 74, 322
displaying in Memory window, 303-304
Source window, 376
setting tab stops, 35
toggling the display mode, 303-304
SRCPATH environment variable, 76
startup options, 32
state qualifier, 127, 213-215, 439-444
status
register, editing, 301
specification for tracing, 395-397
status values, 439-444
for making custom mnemonics, 397
predefined, 389-390
Status window, 379
step
multiple lines, 14
one line, 13
over, interrupting, 354
store conditions, 227-229, 439-444
strings
displaying symbols containing, 86, 317
searching memory for, 108, 288
searching source files, 78, 306
structures (C operators), 159
subnet mask, 414
subroutines, stepping over, 187

Symbol window, 382
copying information, 316-317
searching for strings, 317
symbols, 156
syntax of addresses entered as constants, 401
system setup, 410

T tab stops
settings, 264
setting in the Source window, 35
target system, 439-444
interrupts, 386-388
enabling or disabling interrupts, 49
copying memory into emulation memory, 106, 292
task switch messages, enabling, 244-246
tasks running under Windows, number of, 350
TCP service ports, 414
telnet, 414, 420
Terminal Interface commands, 346
text, selecting, 28
too many instructions execute in single step, 402-403
top of screen commands, 162
trace
foreground/background operation, 266-267
patterns, 130, 134, 216-223, 227-231
range, 232-233
settings, 227-229
signals, 389-390
trace foreground monitor
enabling or disabling, 53
trace messages
enabling or disabling, 52
trace specifications, 395-397
copying, 328
editing, 139, 212
loading, 142
specifying the destination, 328
storing, 141
trace state, 439-444
searching for in Trace Window, 327

Trace window, 383
copying information, 325-326
displaying accumulated count information, 323
displaying bus cycles, 322
displaying relative count information, 323
displaying source only, 322
Trace→Again (F7), (ALT, T, A) command, 226
Trace→Edit... (ALT, T, E) command, 212
Trace→Find Then Trigger... (ALT, T, D) command, 216-219
Trace→Function Caller... (ALT, T, C) command, 206-207
Trace→Function Statement... (ALT, T, S) command, 208-209
Trace→Halt (ALT, T, H) command, 225
Trace→Sequence... (ALT, T, Q) command, 220-223
Trace→Trigger Store... (ALT, T, T) command, 213-215
Trace→Until Halt (ALT, T, U) command, 224
Trace→Variable Access... (ALT, T, V) command, 210-211
transfer address, 12, 89, 91, 184-185, 188-190, 439-444
translating addresses, 254-257
implications, 62-63
trigger, 439-444
condition, 439-444
position, 439-444
state, searching for in Trace window, 326
store condition, 439-444
tutorial, 4
type of memory, 60, 247-250

U
unary minus operator, 159
unexpected stepping behavior, 402-403
unions (C operators), 159
unused prefetches, 391-394
user ID, 421-423
user programs, loading, 73
user-defined symbols
creating, 84, 318
deleting, 85, 320
displaying, 85, 316
Utilities→Copy... (ALT, -, U, C) command, 290
Utilities→Fill... (ALT, -, U, F) command, 291
Utilities→Image... (ALT, -, U, I) command, 292

Utilities→Load... (ALT, -, U, L) command, 294
Utilities→Store... (ALT, -, U, S) command, 295

V
values
  searching memory for, 108, 288
  status, 389-390
variable access breakpoints, 199-202
variable modify breakpoints, 199-202
Variable→Edit... (ALT, V, E) command, 203-204
variables
  auto, 99-101
  displaying, 15, 99
  dynamic, 203-204, 329, 385
  editing, 16, 100, 203-205
  environment, 76
  global, 81, 122, 312
  local, 82-83, 313
  monitoring in the WatchPoint window, 17, 101
  tracing accesses, 23, 122, 210-211
version information, 273, 429
virtual address: definition, 399

W
WAIT command, 152
wait delays, inserting in command files, 347
watchpoints, 439-444
  editing, 329
WatchPoint window, 385
  monitoring variables in, 17, 101
  turning polling ON or OFF, 69
window contents, copying to the list file, 34
Window→1-9 (ALT, W, 1-9) command, 271
Window→Arrange Icons (ALT, W, A) command, 270
Window→Cascade (ALT, W, C) command, 270
Window→More Windows... (ALT, W, M) command, 272
Window→Tile (ALT, W, T) command, 270
windows (debugger), opening, 271-272
Windows for Workgroups LAN address, 352
windows of program execution, tracing, 139
WINSOCK LAN address, 353
writes to ROM, enabling or disabling breaks on, 51

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

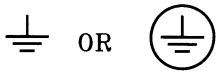
Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

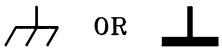
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

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.