User's Guide

Real-Time C Debugger for
68360

Notice

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   B3627-97000, June 1994
Edition 2   B3627-97001, November 1994
Edition 3   B3627-97002, January 1996

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 also controls HP E3490A Software Probes on the LAN. 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 (emulator only)

- 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 68360. It is organized into five parts:

Part 1. Quick Start Guide

Part 2. User's Guide

Part 3. Reference

Part 4. Concept Guide

Part 5. Installation Guide

Contents

2 Getting Started with an HP E3490A Software Probe

10 Window Control Menu Commands

12 Other Command File and Macro Commands

13 Error Messages

Part 1

Quick Start Guide

A few task instructions to help you get comfortable.

Part 1

1
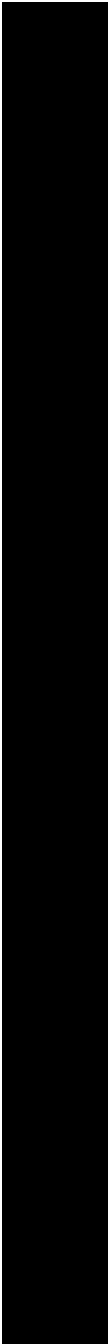
Getting Started with an Emulator

Getting Started with an Emulator

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   Load the demo program.
5   Display the source file.
6   Set a breakpoint.
7   Run the demo program.
8   Delete the breakpoint.
9   Single-step one line.
10 Single-step 10 lines.
11 Display a variable.
12 Edit a variable.
13 Monitor a variable in the WatchPoint window.
14 Run until return from current function.
15 Step over a function.
16 Run the program to a specified line.
17 Display register contents.
18 Trace function flow.
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\M360\DEMO directory (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

Subdirectories exist for the SAMPLE demo program, which is a simple C program that does case conversion on a couple strings, and for the ECS demo

program, which is a somewhat more complex C program for an environmental control system.

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

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

Step 1. Start the debugger

- Open the HP Real-Time C Debugger group box and double-click the 68360 debugger icon.

  Or:

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

2 Enter the debugger startup command, C:\HP\RTC\M360\B3627.EXE (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

3 Choose the OK button.

Step 2. Adjust the fonts and window size

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

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

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

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

The sizes of the RTC window, as well as the sizes of the windows within RTC, and the fonts used will be saved in the B3627.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 program reserves addresses 0h-0fffh for ROM and 6000h-0ffffh for RAM. Map these address ranges as emulation 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 "0fff".

4 Select "erom" in the Type option box.

5 Choose the Apply button.

6 Enter "6000" in the Start text box and "0ffff" in the End text box.

7 Select "eram" in the Type option box.

8 Choose the Apply button.
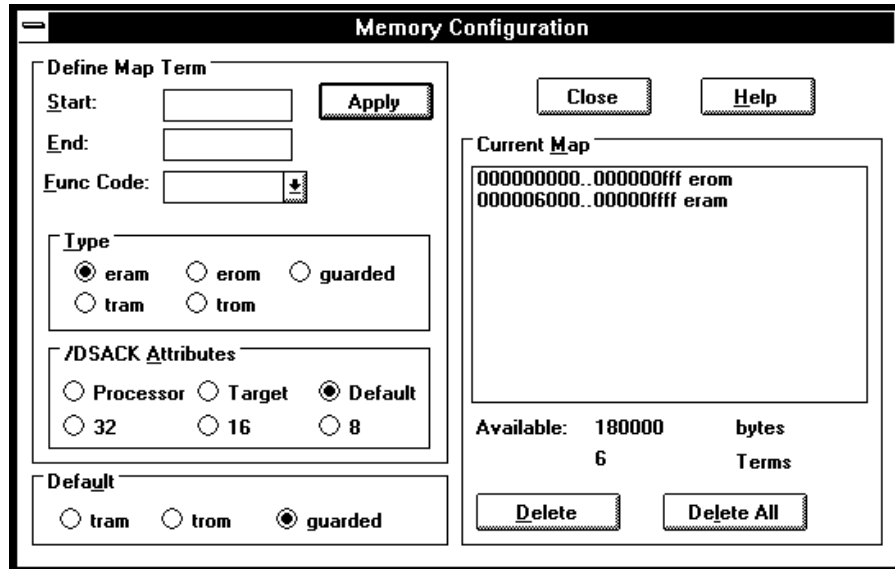
9 Choose the Close button.
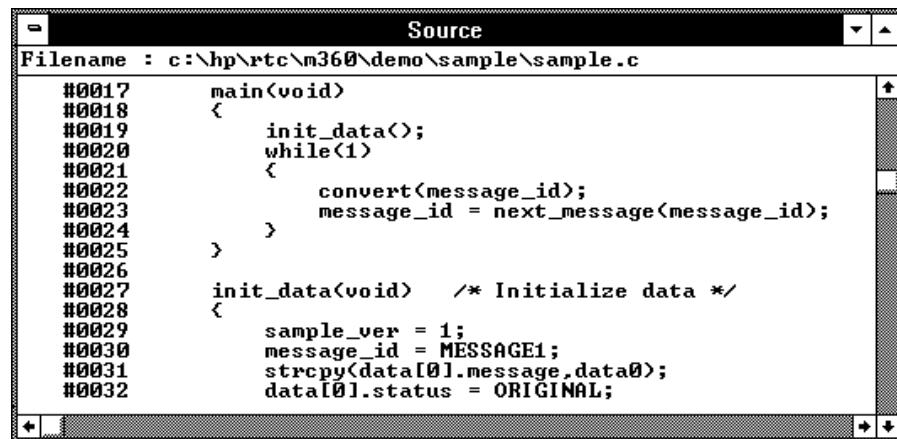
Step 4. Load the demo program

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

2 Choose the Browse button and select the sample program object file, C:\HP\RTC\M360\DEMO\SAMPLE\SAMPLE.X (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

3 Choose the OK button in the Object File Name dialog box.

4 Choose the Load button.

Step 5. 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.

```
┌──────────────────────────────────────────────────────────────────┐
│  ▱                           Source                          ▼ ▲  │
├──────────────────────────────────────────────────────────────────┤
│ Filename : c:\hp\rtc\m360\demo\sample\sample.c                    │
│     #0017        main(void)                                    ▲  │
│     #0018        {                                                │
│     #0019            init_data();                                 │
│     #0020            while(1)                                     │
│     #0021            {                                            │
│     #0022                convert(message_id);                  ▯  │
│     #0023                message_id = next_message(message_id);   │
│     #0024            }                                            │
│     #0025        }                                                │
│     #0026                                                         │
│     #0027        init_data(void)    /* Initialize data */         │
│     #0028        {                                                │
│     #0029            sample_ver = 1;                              │
│     #0030            message_id = MESSAGE1;                        │
│     #0031            strcpy(data[0].message,data0);               │
│     #0032            data[0].status = ORIGINAL;               ▼  │
│ ◆ ▯                                                           ➤ ↓ │
└──────────────────────────────────────────────────────────────────┘
```

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

Step 6. Set a breakpoint

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

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

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

```
 ─                          Source                        ▼ ▲
Filename : c:\hp\rtc\m360\demo\sample\sample.c
      #0017        main(void)                                  ✦
      #0018        {
      #0019            init_data();
      #0020            while(1)
      #0021            {
 BP   #0022   |          convert(message_id);
      #0023               message_id = next_message(message_id);
      #0024            }
      #0025        }
      #0026
      #0027        init_data(void)    /* Initialize data */
      #0028        {
      #0029            sample_ver = 1;
      #0030            message_id = MESSAGE1;
      #0031            strcpy(data[0].message,data0);
      #0032            data[0].status = ORIGINAL;
 ←                                                          → ↓
```

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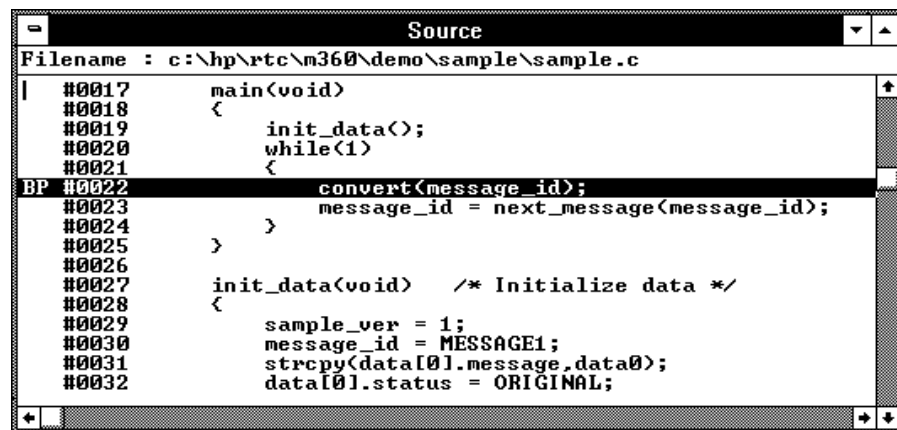
Note

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

Step 7. Run the demo program

To run the demo program from the transfer address:

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

2 Select the Start Address option.

3 Choose the Run button.

```
┌─────────────────────────────────────────────────────────────────┐
│ ▭                         Source                          ▼ ▲ │
├─────────────────────────────────────────────────────────────────┤
│ Filename : c:\hp\rtc\m360\demo\sample\sample.c                  │
├─────────────────────────────────────────────────────────────────┤
│ │   #0017      main(void)                                     ▲ │
│     #0018      {                                                │
│     #0019          init_data();                                 │
│     #0020          while(1)                                     │
│     #0021          {                                          █ │
│ BP  #0022              convert(message_id);                     │
│     #0023              message_id = next_message(message_id);   │
│     #0024          }                                            │
│     #0025      }                                                │
│     #0026                                                       │
│     #0027      init_data(void)    /* Initialize data */         │
│     #0028      {                                                │
│     #0029          sample_ver = 1;                              │
│     #0030          message_id = MESSAGE1;                       │
│     #0031          strcpy(data[0].message,data0);               │
│     #0032          data[0].status = ORIGINAL;                   │
├─────────────────────────────────────────────────────────────────┤
│ ◄ █                                                         ► ▼ │
└─────────────────────────────────────────────────────────────────┘
```

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

Step 8. Delete the breakpoint

To delete the breakpoint set on line 22:

1 Cursor-select line 22.

2 Choose the Breakpoint→Delete at Cursor (ALT, B, D) command.

The "BP" marker disappears in the Source window.

Step 9. 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 10. 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.
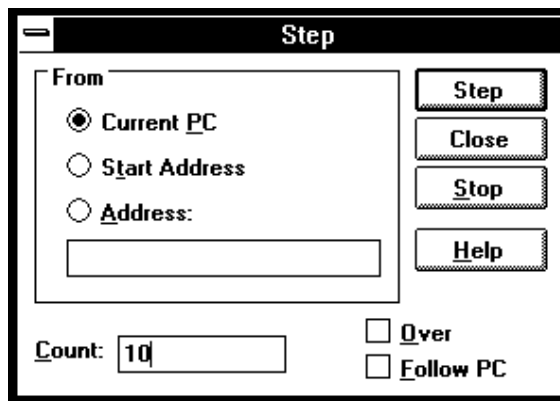
4 Choose the Step button. Notice that the step count decrements by one as the program executes step by step. The step count stops at 0.

5 Choose the Close button.

Step 11. Display a variable

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

1 Drag "*mes" on line 45 in the Source window until it is highlighted.

2 Choose the Variable→Edit... (ALT, V, E) command.
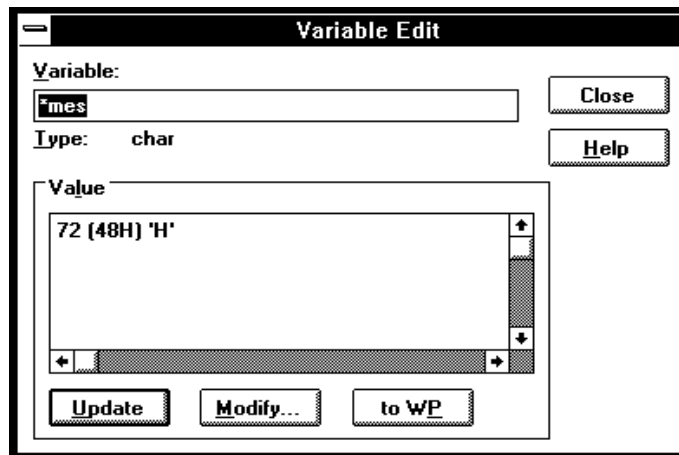
The Variable text box displays "*mes".

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

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

Step 12. 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.
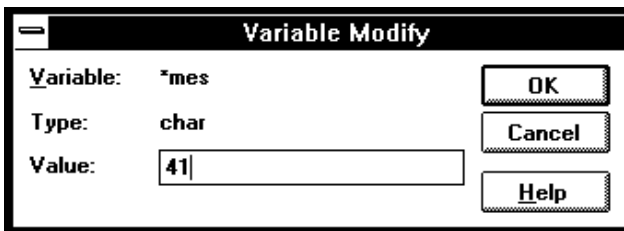
3 Choose the OK button.

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

Step 13. 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.
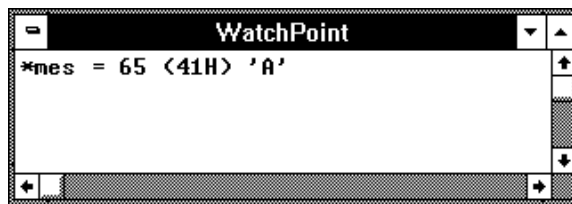
Notice the variable "*mes" has been registered as a watchpoint.

Step 14. Run until return from current function

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

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

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

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

Step 15. Step over a function

To step over "change_status":
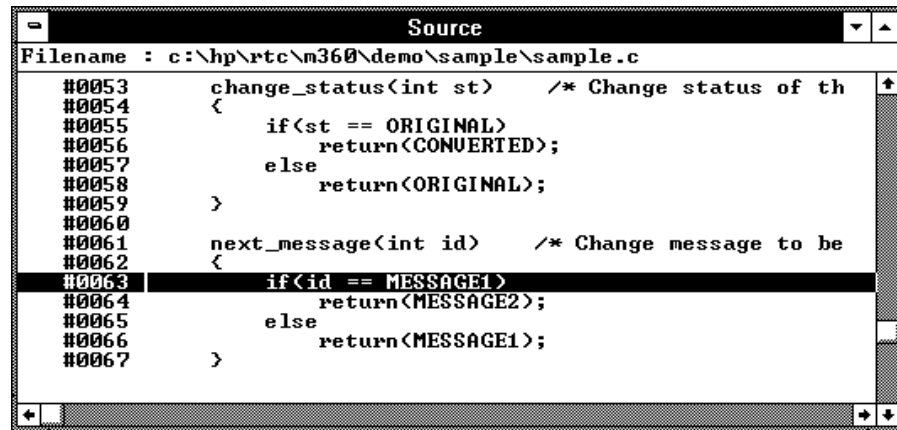
- Choose the Execution→Step Over (ALT, E, O) command. Or, press the F3 key.

The "change_status" function executes, and line 41 is highlighted to show that it is the next line which will be executed.

Step 16. Run the program to a specified line

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

1 Cursor-select line 63.

2 Choose the Execution→Run to Cursor (ALT, E, C) command.
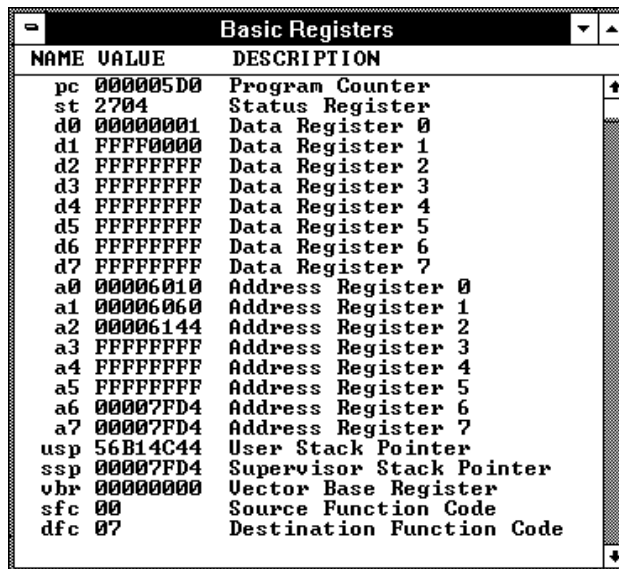
```
 ▭                           Source                        ▾ ▴
Filename : c:\hp\rtc\m360\demo\sample\sample.c
    #0053        change_status(int st)    /* Change status of th    ♦
    #0054        {
    #0055            if(st == ORIGINAL)
    #0056                return(CONVERTED);
    #0057            else
    #0058                return(ORIGINAL);
    #0059        }
    #0060
    #0061        next_message(int id)     /* Change message to be
    #0062        {
    #0063 |          if(id == MESSAGE1)
    #0064                return(MESSAGE2);
    #0065            else
    #0066                return(MESSAGE1);
    #0067        }

 ← ▯                                                       → ↓
```

The program executes and stops immediately before line 63.

Step 17. Display register contents

1 Choose the Window→Basic Registers command.

```
┌─┬──────────────────────────────────────────┬─┬─┐
│ ▬ │            Basic Registers            │▼│▲│
├──────────────────────────────────────────────┤
│ NAME VALUE     DESCRIPTION                    │
│  pc 000005D0   Program Counter            │▲│ │
│  st 2704       Status Register                │
│  d0 00000001   Data Register 0                │
│  d1 FFFF0000   Data Register 1                │
│  d2 FFFFFFFF   Data Register 2                │
│  d3 FFFFFFFF   Data Register 3                │
│  d4 FFFFFFFF   Data Register 4                │
│  d5 FFFFFFFF   Data Register 5                │
│  d6 FFFFFFFF   Data Register 6                │
│  d7 FFFFFFFF   Data Register 7                │
│  a0 00006010   Address Register 0             │
│  a1 00006060   Address Register 1             │
│  a2 00006144   Address Register 2             │
│  a3 FFFFFFFF   Address Register 3             │
│  a4 FFFFFFFF   Address Register 4             │
│  a5 FFFFFFFF   Address Register 5             │
│  a6 00007FD4   Address Register 6             │
│  a7 00007FD4   Address Register 7             │
│ usp 56B14C44   User Stack Pointer             │
│ ssp 00007FD4   Supervisor Stack Pointer       │
│ vbr 00000000   Vector Base Register           │
│ sfc 00         Source Function Code           │
│ dfc 07         Destination Function Code  │▼│ │
└──────────────────────────────────────────────┘
```

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

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

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

```
┌──────────────────────────────────────────────────────┐
│ ■              Basic Registers              ▼ ▲       │
│ NAME VALUE        DESCRIPTION                          │
│   pc --------   Program Counter                   ↑   │
│   st --------   Status Register                   ▓   │
│   d0 --------   Data Register 0                    ▓   │
│   d1 --------   Data Register 1                    ▓   │
│   d2 --------   Data Register 2                    ▓   │
│   d3 --------   Data Register 3                    ▓   │
│   d4 --------   Data Register 4                    ▓   │
│   d5 --------   Data Register 5                    ▓   │
│   d6 --------   Data Register 6                    ▓   │
│   d7 --------   Data Register 7                    ▓   │
│   a0 --------   Address Register 0                 ▓   │
│   a1 --------   Address Register 1                 ▓   │
│   a2 --------   Address Register 2                 ▓   │
│   a3 --------   Address Register 3                 ▓   │
│   a4 --------   Address Register 4                 ▓   │
│   a5 --------   Address Register 5                 ▓   │
│   a6 --------   Address Register 6                 ▓   │
│   a7 --------   Address Register 7                 ▓   │
│  usp --------   User Stack Pointer                 ▓   │
│  ssp --------   Supervisor Stack Pointer               │
│  vbr --------   Vector Base Register                   │
│  sfc --------   Source Function Code                   │
│  dfc --------   Destination Function Code          ↓   │
└──────────────────────────────────────────────────────┘
```
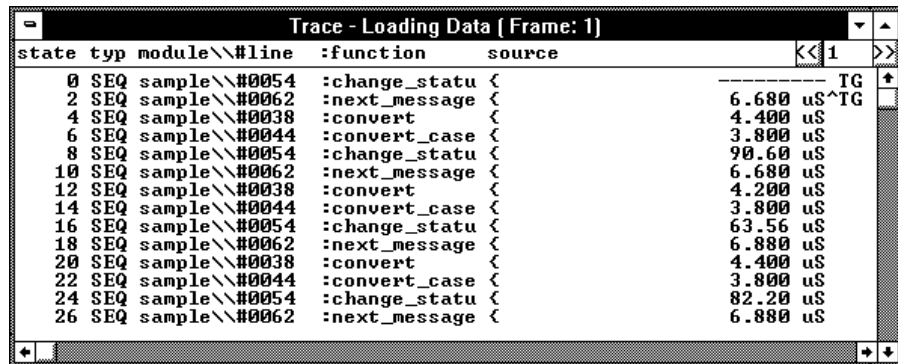
Notice that register contents are replaced with "----" in the display. This shows the debugger cannot update the register display. In order for the emulator to update its register display, the emulation monitor must interrupt target program execution while it reads the registers.

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

Step 18. Trace function flow

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

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

```
▬            Trace - Loading Data [ Frame: 1]                 ▼ ▲
state typ module\\#line   :function      source           <<|1    >>
     0 SEQ sample\\#0054   :change_statu {              ---------- TG   ▲
     2 SEQ sample\\#0062   :next_message {              6.680 uS^TG
     4 SEQ sample\\#0038   :convert      {              4.400 uS
     6 SEQ sample\\#0044   :convert_case {              3.800 uS
     8 SEQ sample\\#0054   :change_statu {              90.60 uS
    10 SEQ sample\\#0062   :next_message {              6.680 uS
    12 SEQ sample\\#0038   :convert      {              4.200 uS
    14 SEQ sample\\#0044   :convert_case {              3.800 uS
    16 SEQ sample\\#0054   :change_statu {              63.56 uS
    18 SEQ sample\\#0062   :next_message {              6.880 uS
    20 SEQ sample\\#0038   :convert      {              4.400 uS
    22 SEQ sample\\#0044   :convert_case {              3.800 uS
    24 SEQ sample\\#0054   :change_statu {              82.20 uS
    26 SEQ sample\\#0062   :next_message {              6.880 uS
◄ ▮                                                            ► ▼
```

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

Step 19. Trace a function's callers
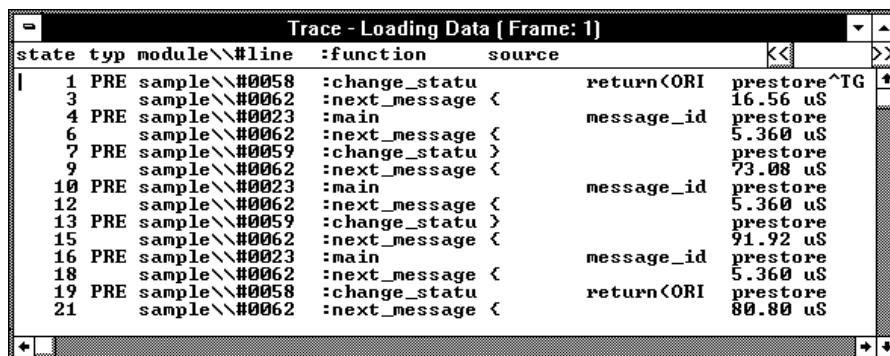
To trace the caller of "next_message":

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

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

```
┌──────────────────────────────────────────────────┐
│ ▬          Function Caller Trace                  │
├──────────────────────────────────────────────────┤
│ Function:                        ┌─────────┐      │
│                                  │   OK    │      │
│ ┌────────────────────────────┐   └─────────┘      │
│ │next_message                │   ┌─────────┐      │
│ └────────────────────────────┘   │ Cancel  │      │
│                                  └─────────┘      │
│                                  ┌─────────┐      │
│                                  │  Help   │      │
│                                  └─────────┘      │
└──────────────────────────────────────────────────┘
```
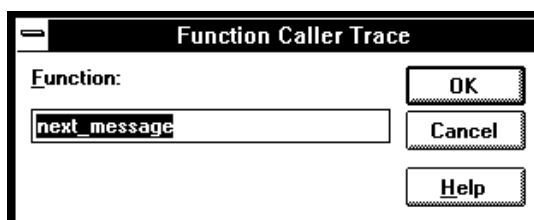
3 Choose the OK button.

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

```
┌─────────────────────────────────────────────────────────────────────────┐
│ ▬           Trace - Loading Data [ Frame: 1]                    ▼ ▲       │
├─────────────────────────────────────────────────────────────────────────┤
│state typ module\\#line   :function        source            K<       >>  │
│    1 PRE sample\\#0058    :change_statu    return(ORI    prestore^TG ↑    │
│    3     sample\\#0062    :next_message <                 16.56 uS        │
│    4 PRE sample\\#0023    :main            message_id    prestore        │
│    6     sample\\#0062    :next_message <                 5.360 uS        │
│    7 PRE sample\\#0059    :change_statu }                 prestore        │
│    9     sample\\#0062    :next_message <                 73.08 uS        │
│   10 PRE sample\\#0023    :main            message_id    prestore        │
│   12     sample\\#0062    :next_message <                 5.360 uS        │
│   13 PRE sample\\#0059    :change_statu }                 prestore        │
│   15     sample\\#0062    :next_message <                 91.92 uS        │
│   16 PRE sample\\#0023    :main            message_id    prestore        │
│   18     sample\\#0062    :next_message <                 5.360 uS        │
│   19 PRE sample\\#0058    :change_statu    return(ORI    prestore        │
│   21     sample\\#0062    :next_message <                 80.80 uS        │
│ ←▮                                                              →↓       │
└─────────────────────────────────────────────────────────────────────────┘
```

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

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

Step 20. Trace access to a variable
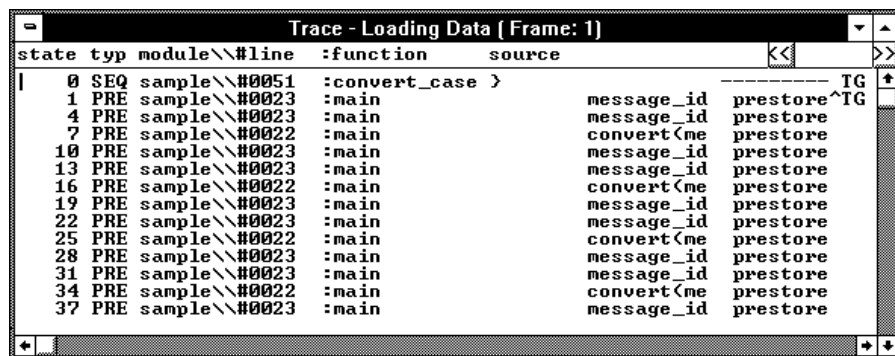
To trace access to variable "message_id":

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

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

```
╔══════════ Variable Access Trace ══════════╗
║ Variable:                    ┌─────────┐   ║
║                              │   OK    │   ║
║ ┌──────────────────────────┐ └─────────┘   ║
║ │ message_id               │ ┌─────────┐   ║
║ └──────────────────────────┘ │ Cancel  │   ║
║                              └─────────┘   ║
║                              ┌─────────┐   ║
║                              │  Help   │   ║
║                              └─────────┘   ║
╚════════════════════════════════════════════╝
```
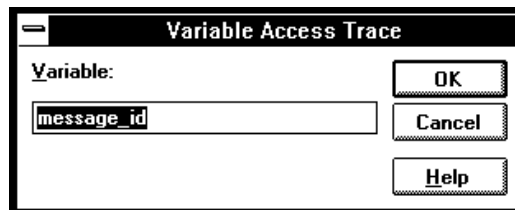
3 Choose the OK button.

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

```
┌─────────────── Trace - Loading Data [ Frame: 1] ───────────────┐
│state typ module\\#line   :function      source           |<|    >>|
│    0 SEQ sample\\#0051    :convert_case >              ---------- TG
│    1 PRE sample\\#0023    :main              message_id  prestore^TG
│    4 PRE sample\\#0023    :main              message_id  prestore
│    7 PRE sample\\#0022    :main              convert(me   prestore
│   10 PRE sample\\#0023    :main              message_id  prestore
│   13 PRE sample\\#0023    :main              message_id  prestore
│   16 PRE sample\\#0022    :main              convert(me   prestore
│   19 PRE sample\\#0023    :main              message_id  prestore
│   22 PRE sample\\#0023    :main              message_id  prestore
│   25 PRE sample\\#0022    :main              convert(me   prestore
│   28 PRE sample\\#0023    :main              message_id  prestore
│   31 PRE sample\\#0023    :main              message_id  prestore
│   34 PRE sample\\#0022    :main              convert(me   prestore
│   37 PRE sample\\#0023    :main              message_id  prestore
└────────────────────────────────────────────────────────────────┘
```

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

If you encounter problems when using the emulator/analyzer, refer to the chapter titled "Solving Problems" in the MC68360 Emulator/Analyzer Installation/Service/Terminal Interface Manual.

2

Getting Started with an HP E3490A
Software Probe

Getting Started with an HP E3490A Software Probe

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 Load the demo program and configure initial register values.
3 Display the source file.
4 Set a breakpoint.
5 Run the demo program.
6 Delete the breakpoint.
7 Display a variable.
8 Edit a variable.
9 Monitor a variable in the WatchPoint window.
10 Single-step one line.
11 Run until return from current function.
12 Step over a function.
13 Run the program to a specified line.
14 Display register contents.
15 Exit the debugger.

Demo Programs

Demo programs are included with the Real-Time C Debugger in the C:\HP\RTC\M360\DEMO\SFTPROBE directory (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

This tutorial shows you how to perform some measurements on the ECSMON demo program. The ECS demo program is a somewhat complex C program for an environmental control system.

Demo Target System

You must connect the HP E3490A Software Probe to a target system before
you run the demo program. If your target system hardware is not working
yet, obtain an evaluation board such as an EST Single Board Computer.

Step 1. Start the debugger

- Open the HP Real-Time C Debugger group box and double-click the 68360 debugger icon.

  Or:

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

2 Enter the debugger startup command, C:\HP\RTC\M360\B3627.EXE (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

3 Choose the OK button.

Step 2. Load the demo program and configure initial register values

Before you can load a program, you must run your target system's initialization code, then set the User Stack Pointer. To simplify this process, a configuration file is supplied with the demo program.

1 Choose the File→Load Emulator Config... (ALT, F, E) command.

```
File  Execution  Breakpoint  Variable
 Load Object...
 Flash Programming...
 Command Log              ▶
 Run Cmd File...

 Load Debug...
 Save Debug...

 Load Emulator Config...
 Save Emulator Config...

 Copy Destination...

 Exit
 Exit HW Locked
```
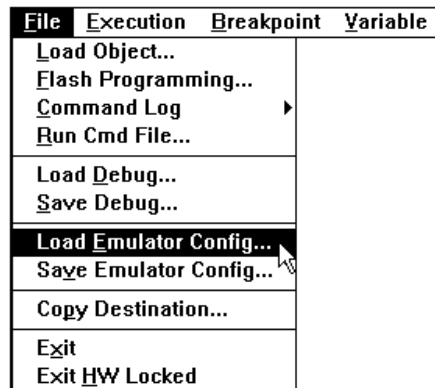
2 Select the sample configuration command file, C:\HP\RTC\M360\DEMO\SFTPROBE\ECSMON.CMD (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

```
┌─────────────────────────────────────────────────────────────┐
│ ▬          Load Emulator Configuration                       │
├─────────────────────────────────────────────────────────────┤
│  File Name:                  Directories:         ┌─────────┐│
│  ┌──────────────────────┐    c:\...\m360\demo\sftprobe│  OK   ││
│  │ ecsmon.cmd           │                         └─────────┘│
│  ┌──────────────────────┐    ┌─────────────────┐  ┌─────────┐│
│  │ ecsmon.cmd           │█   │ 🗀 c:\          │  │ Cancel  ││
│  │                      │    │  🗀 hp          │  └─────────┘│
│  │                      │    │   🗀 rtc        │  ┌─────────┐│
│  │                      │    │    🗀 m360      │  │ Help    ││
│  │                      │    │     🗀 demo     │  └─────────┘│
│  │                      │    │      🗁 sftprobe│            │
│  └──────────────────────┘    └─────────────────┘            │
│                                                              │
│  List Files of Type:          Drives:                        │
│  ┌───────────────────┬──┐    ┌───────────────────────┬──┐    │
│  │ Command Files(*.CMD)│ ± │    │ ▭ c: ms-dos_5        │ ± │    │
│  └───────────────────┴──┘    └───────────────────────┴──┘    │
└─────────────────────────────────────────────────────────────┘
```
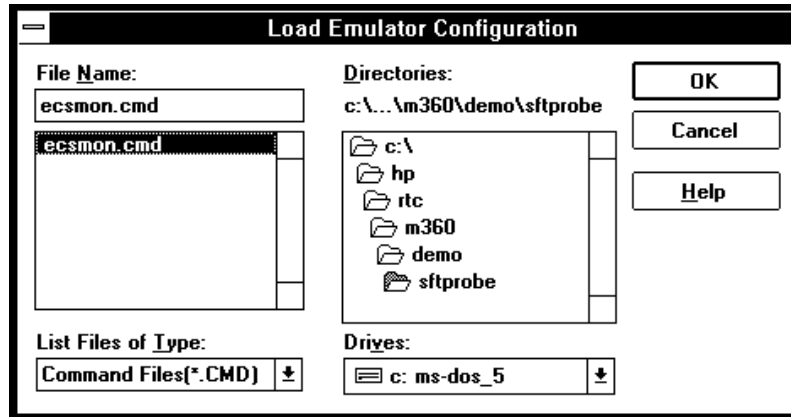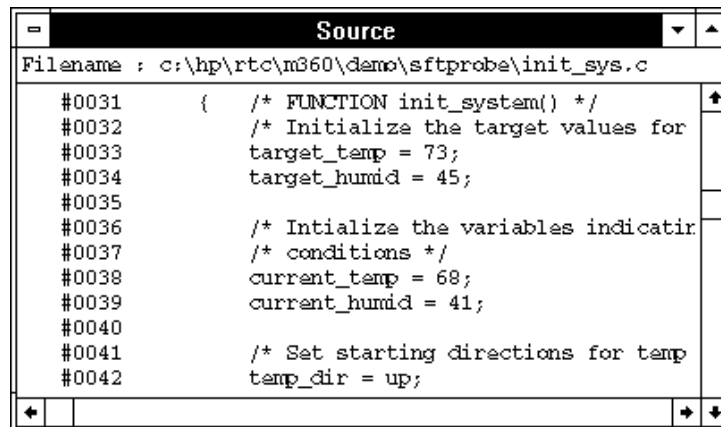
3 Choose the OK button in the Load Emulator Configuration dialog box.

This configuration command file will run your target system's built-in
initialization code, break, set the User Stack Pointer, and then load the object
file for the demo program, ECSMON.X.

Step 3. Display the source file

To display the INIT_SYS.C source file starting from the function init_system:

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 "init_system." You may need to scroll down to see "init_system."

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.
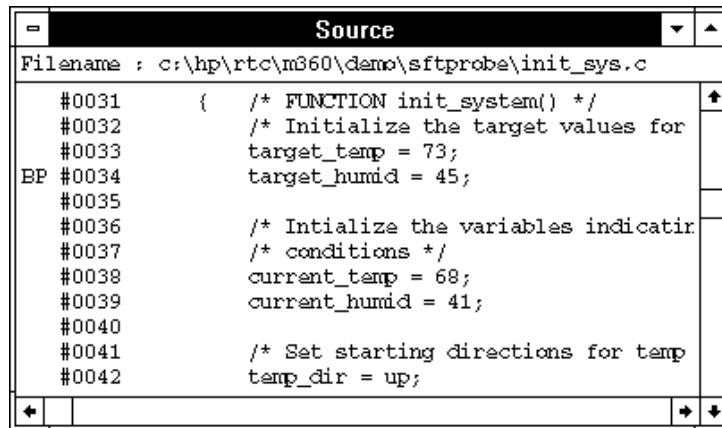
```
 ═                        Source                      ▼  ▲
Filename : c:\hp\rtc\m360\demo\sftprobe\init_sys.c
      #0031      {    /* FUNCTION init_system() */       ✦
      #0032            /* Initialize the target values for
      #0033            target_temp = 73;
      #0034            target_humid = 45;
      #0035
      #0036            /* Intialize the variables indicatin
      #0037            /* conditions */
      #0038            current_temp = 68;
      #0039            current_humid = 41;
      #0040
      #0041            /* Set starting directions for temp
      #0042            temp_dir = up;
 ✦                                                    ➡  ✦
```

The window displays INIT_SYS.C source file, starting from init_system function.

Step 4. Set a breakpoint

To set a breakpoint on line 34 in INIT_SYS.C:

1 Cursor-select line 34 (that is, move the mouse pointer over line 34 and click the left mouse button).

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

```
┌─────────────────────────────────────────────────────────┐
│ ▬              Source                           ▼  ▲     │
├─────────────────────────────────────────────────────────┤
│ Filename : c:\hp\rtc\m360\demo\sftprobe\init_sys.c       │
├─────────────────────────────────────────────────────────┤
│      #0031      {    /* FUNCTION init_system() */     ▲  │
│      #0032           /* Initialize the target values for │
│      #0033           target_temp = 73;                   │
│ BP   #0034           target_humid = 45;                  │
│      #0035                                               │
│      #0036           /* Intialize the variables indicatin│
│      #0037           /* conditions */                    │
│      #0038           current_temp = 68;                  │
│      #0039           current_humid = 41;                 │
│      #0040                                               │
│      #0041           /* Set starting directions for temp │
│      #0042           temp_dir = up;                      │
│ ◄                                            ►  ▼       │
└─────────────────────────────────────────────────────────┘
```

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

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

Step 5. Run the demo program

To run the demo program from the transfer address:

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

2 Select the Start Address option.

```
┌─────────────────────────────────────────┐
│ ▬               Run                      │
├─────────────────────────────────────────┤
│ ┌─From─────────────────┐   ┌──────────┐  │
│ │                      │   │   Run    │  │
│ │  ○ Current PC        │   └──────────┘  │
│ │                      │   ┌──────────┐  │
│ │  ◉ Start Address     │ �│  Cancel  │  │
│ │                      │   └──────────┘  │
│ │  ○ User Reset        │   ┌──────────┐  │
│ │                      │   │   Help   │  │
│ │  ○ Address:          │   └──────────┘  │
│ │  ┌────────────────┐  │                 │
│ │  └────────────────┘  │                 │
│ └──────────────────────┘                 │
└─────────────────────────────────────────┘
```

3 Choose the Run button.

```
┌────────────────────────────────────────────────────┬───┬───┐
│ ▬                    Source                         │ ▼ │ ▲ │
├────────────────────────────────────────────────────┴───┴───┤
│ Filename : c:\hp\rtc\m360\demo\sftprobe\init_sys.c         │
├──────────────────────────────────────────────────────┬───┤
│ │    #0031       {    /* FUNCTION init_system() */    │ ▲ │
│      #0032            /* Initialize the target values for│   │
│      #0033            target_temp = 73;                │   │
│ BP   #0034            target_humid = 45;   I           │   │
│      #0035                                             │   │
│      #0036            /* Intialize the variables indicatin│ │
│      #0037            /* conditions */                 │   │
│      #0038            current_temp = 68;               │   │
│      #0039            current_humid = 41;              │   │
│      #0040                                             │   │
│      #0041            /* Set starting directions for temp│  │
│      #0042            temp_dir = up;                   │ ▼ │
├──────────────────────────────────────────────────────┼───┤
│ ◄ │                                            │ ► │ ▼ │
└────────────────────────────────────────────────────────────┘
```
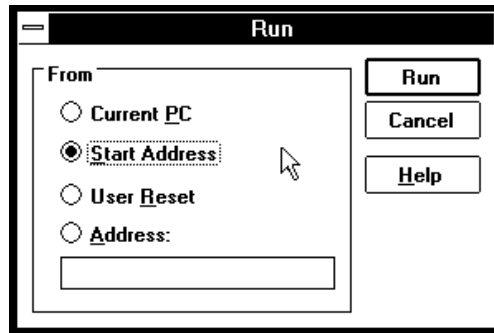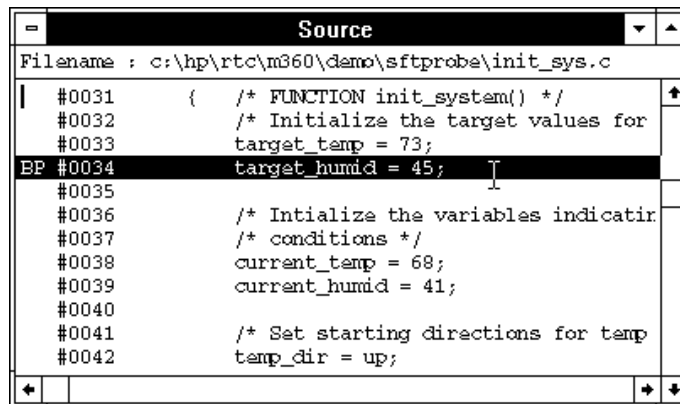
Notice the demo program runs until line 34. The highlighted line indicates the current program counter.

Step 6. 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 7. Display a variable

To display the contents of variable "target_temp":

1 Drag "target_temp" on line 33 in the Source window until it is highlighted.

2 Choose the Variable→Edit... (ALT, V, E) command.
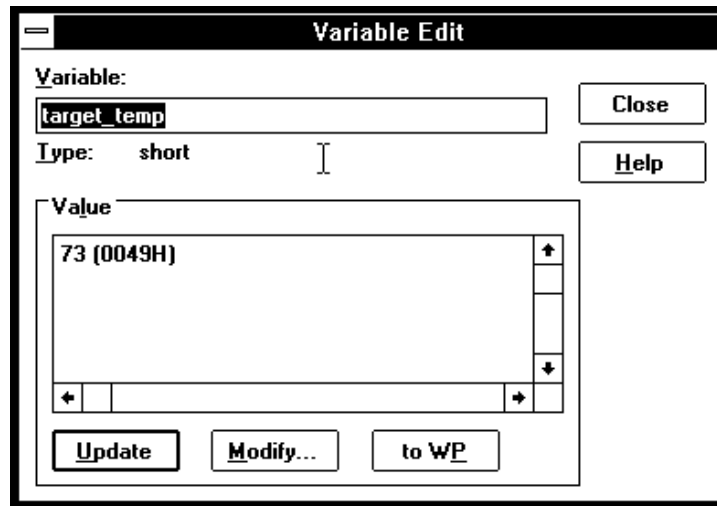
The Variable text box displays "target_temp."

Notice the Value list box displays the contents of "target_temp."

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 8. Edit a variable

To edit the contents of variable "target_temp":

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

2 Enter "74t" in the Value text box. The "t" indicates that you are entering a decimal value.
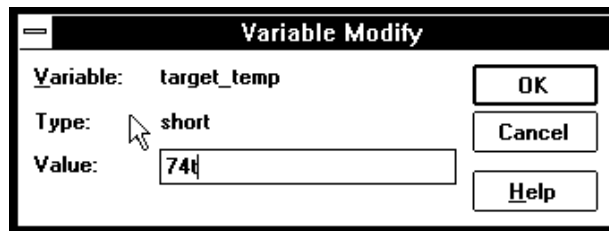
3 Choose the OK button.

4 Notice the contents of the variable in the Value list box has changed to "74."

Step 9. 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 "target_temp" 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.
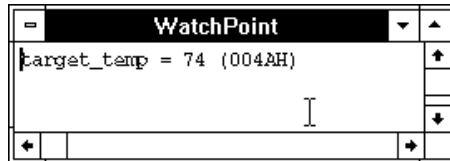
Notice the variable "target_temp" has been registered as a watchpoint.

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.

The next source line will be highlighted, showing that one source line has been executed.

Step 11. Run until return from current function

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

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

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

Step 12. Step over a function

To step over "proc_spec_init":
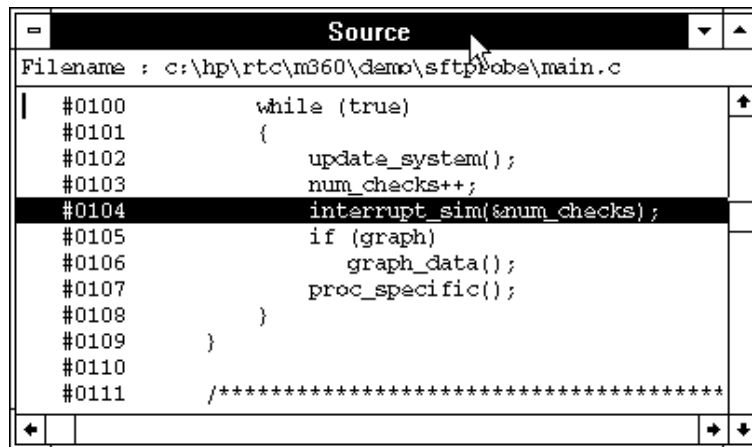
- Choose the Execution→Step Over (ALT, E, O) command. Or, press the F3 key.

The command executes the "proc_spec_init" function.

Step 13. Run the program to a specified line

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

1 Cursor-select line 104.

2 Choose the Execution→Run to Cursor (ALT, E, C) command.
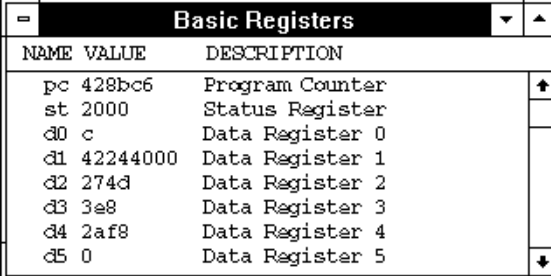
```
┌──────────────────────────────────────────────────────┐
│ ▬                    Source              ▲      ▼   ▲ │
├──────────────────────────────────────────────────────┤
│ Filename : c:\hp\rtc\m360\demo\sftprobe\main.c         │
├──────────────────────────────────────────────────────┤
│┃   #0100          while (true)                      ↑ │
││   #0101          {                                   │
││   #0102              update_system();                │
││   #0103              num_checks++;                   │
││   #0104              interrupt_sim(&num_checks);    │
││   #0105              if (graph)                      │
││   #0106                  graph_data();               │
││   #0107              proc_specific();                │
││   #0108          }                                   │
││   #0109      }                                       │
││   #0110                                              │
││   #0111      /****************************************│
├──────────────────────────────────────────────────────┤
│ ←  │                                          │  →  ↓ │
└──────────────────────────────────────────────────────┘
```

The program executes and stops immediately before line 104.

Step 14. Display register contents
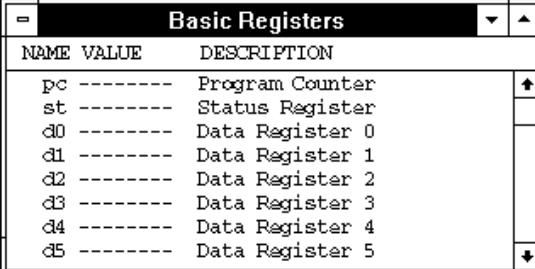
1 Choose the Window→Basic Registers command.

```
┌──────────────────────────────────────────────┐
│ ▬        Basic Registers          ▼  ▲        │
├──────────────────────────────────────────────┤
│ NAME VALUE       DESCRIPTION                   │
│   pc 428bc6      Program Counter          ✦    │
│   st 2000        Status Register               │
│   d0 c           Data Register 0               │
│   d1 42244000    Data Register 1               │
│   d2 274d        Data Register 2               │
│   d3 3e8         Data Register 3               │
│   d4 2af8        Data Register 4               │
│   d5 0           Data Register 5          ✦    │
└──────────────────────────────────────────────┘
```

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

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

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

```
┌──────────────────────────────────────────────┐
│ ▬        Basic Registers          ▼  ▲        │
├──────────────────────────────────────────────┤
│ NAME VALUE       DESCRIPTION                   │
│   pc --------    Program Counter          ✦    │
│   st --------    Status Register               │
│   d0 --------    Data Register 0               │
│   d1 --------    Data Register 1               │
│   d2 --------    Data Register 2               │
│   d3 --------    Data Register 3               │
│   d4 --------    Data Register 4               │
│   d5 --------    Data Register 5          ✦    │
└──────────────────────────────────────────────┘
```

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

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

Step 15. 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.

Part 2

3
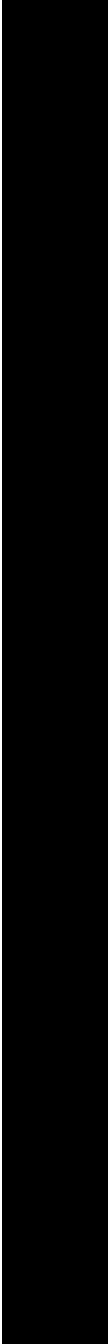
Using the Debugger Interface

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).
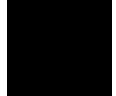
Debugger Function Key Definitions

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
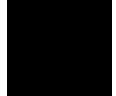
To start the debugger

- 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\M360\B3627.EXE (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).

3 Choose the OK button.

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

To exit the debugger

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

2 Choose the OK button.

This will end your Real-Time C Debugger session.

To create an icon for a different emulator

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

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

3 Select the Program Item option and choose OK.

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

5 In the Command Line text box, enter the "C:\HP\RTC\M360\B3627.EXE -T<transport> -E<connectname>" command (if C:\HP\RTC\M360 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 B3627.INI file.

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

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

6 In the Working Directory text box, enter the directory that contains the debugger program (for example, C:\HP\RTC\M360).

7 Choose the OK button.

Working with Debugger Windows

This section shows you how:

- To open debugger windows

- To copy window contents to the list file

- To change the list file destination

- To change the debugger window fonts

- To set tab stops in the Source window

- To set colors in the Source window

To open debugger windows

- Double-click the icon for the particular window.

- Or, choose the particular window from the Window→ menu.

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

To copy window contents to the list file

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

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

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

To change the list file destination

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

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

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

List file names have the ".LST" extension.

To change the debugger window fonts

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

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

3 Choose the OK button.

To set tab stops in the Source window

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

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

3 Choose the OK button.

The tab width must be between 1 and 20.

To set colors in the Source window

1 Exit the RTC interface and find the initialization file (B3627.INI). It should be in the directory where you installed the RTC product (C:\HP\RTC\, by default).

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

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

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

• The <color> entry may be in upper-case or lower-case letters.

• When ColorMode=ON, these are the default colors:

   • ColorPC=GREEN
   • ColorSource=RED
   • ColorMne=BLUE

• The default color is black if an option is given a null value.

• The options under [Color] set colors as follows:

   • ColorPc sets the color of the line of the current program counter.
   • ColorSource sets the color of the line numbers of source lines.
   • ColorMne sets the color of the address of all mnemonic lines.

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

Using Command Files

This section shows you how:

- To create a command file

- To execute a command file

- To create buttons that execute command files

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

To create a command file

1 Choose the File→Command Log→Log File Name... (ALT, F, C, N) command.

2 Enter the command file name.

3 Choose the File→Command Log→Logging ON (ALT, F, C, O) command.

4 Choose the commands to be stored in the command file.

5 Once the commands have been completed, choose the File→Command Log→Logging OFF (ALT, F, C, F) command.

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

To execute a command file

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

2 Select the command file to be executed.

3 Choose the Execute button.

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

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

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

Example

Command File Being Executed
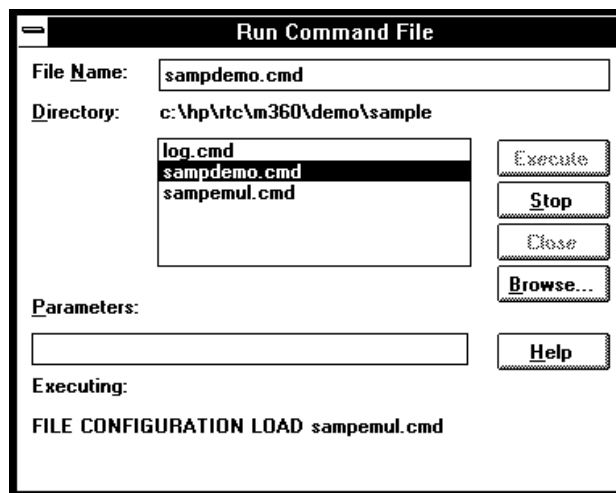
To create buttons that execute command files

1 Activate the Button window by clicking on the Button window icon or by choosing the Window→Button command.

2 From the Button window's control menu, choose the Edit... (ALT, -, E) command.

3 In the Command text box, enter "FILE COMMAND", a space, and the name of the command file to be executed.

4 Enter the button label in the Name text box.

5 Choose the Add button.

6 Choose the Close button.

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

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

4

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 the probe from the demo target system

- Step 3. Select a clock module

- Step 4. Plug the probe into the target system

- Step 5. Turn ON power

- If the emulator doesn't work with the target oscillator

- If the emulator doesn't work with the target crystal

If you are using an HP E3490A Software Probe instead of an emulator, refer to the *HP E3490A Software Probe User's Guide* for information about plugging into your target system.

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 OFF power to the HP 64700 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 OFF power to that target system.

2 Turn OFF power to the emulator.

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. Select a clock module

A clock module must be selected and installed in the emulator probe.

Your initial choice should be to use the clock circuitry from the target system. The emulator comes configured for operation with an oscillator in the target system. This is the Target Oscillator Clock Module, 64780-66509.

64780-66509
Target Oscillator
Clock Module

64780E13
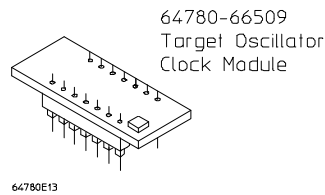
If a crystal circuit is used in the target, Clock Module B from the Clock Module Kit, 64780-66507, should be broken off and used instead.

Clock Module Kit
64780-66507

Module A
Test
Clock Module

Module B
Target Crystal
Clock Module

Module C
4.194 MHz
Clock Module

Module D
User Definable
Clock Module

Module E
32.768 KHz
Clock Module

64780E10
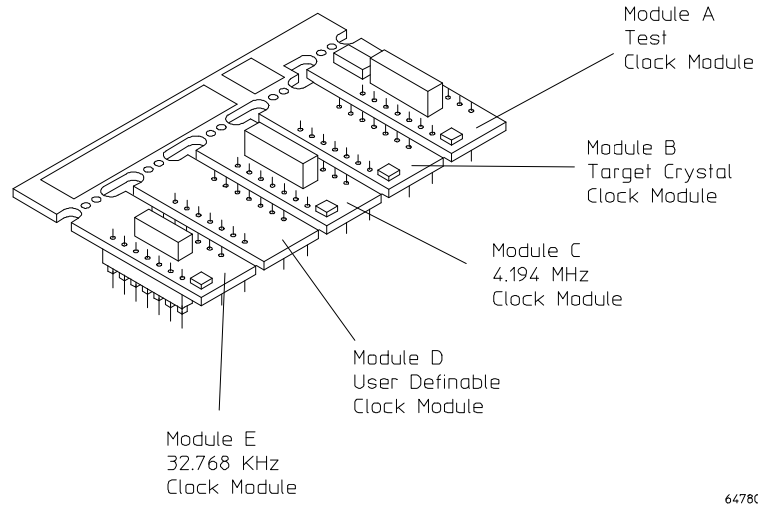
If you can use the Target Oscillator Clock Module, go on to Step 4; otherwise, perform the following steps.

1 Remove plastic rivets that secure the plastic cover on the top of the emulator probe, and remove the cover.

2 Install the appropriate clock module on the emulator probe. Make sure to align pin 1 correctly.

CLOCK MODULE
SOCKET PIN A1

64780E16
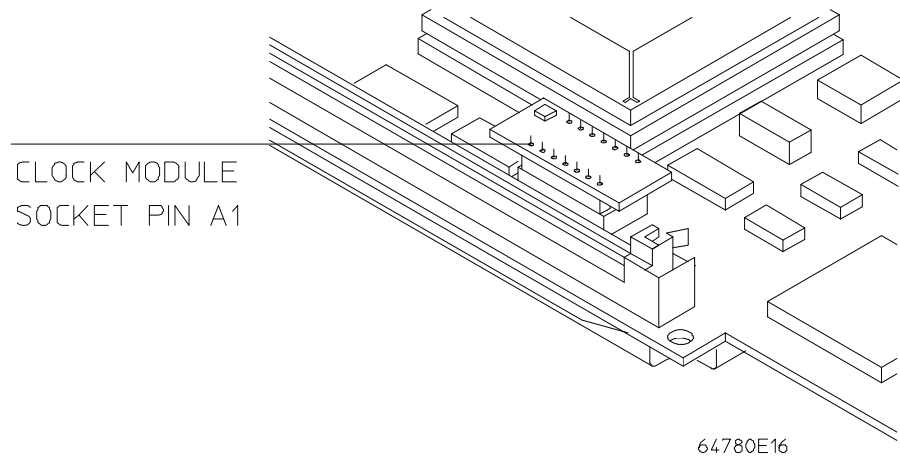
3 Replace the plastic cover, and insert new plastic rivets (supplied with the emulator) to secure the cover.

If you have problems getting the clock to operate correctly, refer to:

- If the emulator doesn't work with the target oscillator

- If the emulator doesn't work with the target crystal

Clock Module A is used for performance verification testing.

For details of selecting a clock module, refer to the chapter titled, "Plugging the emulator into a target system" in the MC68360 Emulator/Analyzer (HP 64780A) Installation/Service/Terminal Interface manual supplied with the product hardware.

Step 4. 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.**

64780E19
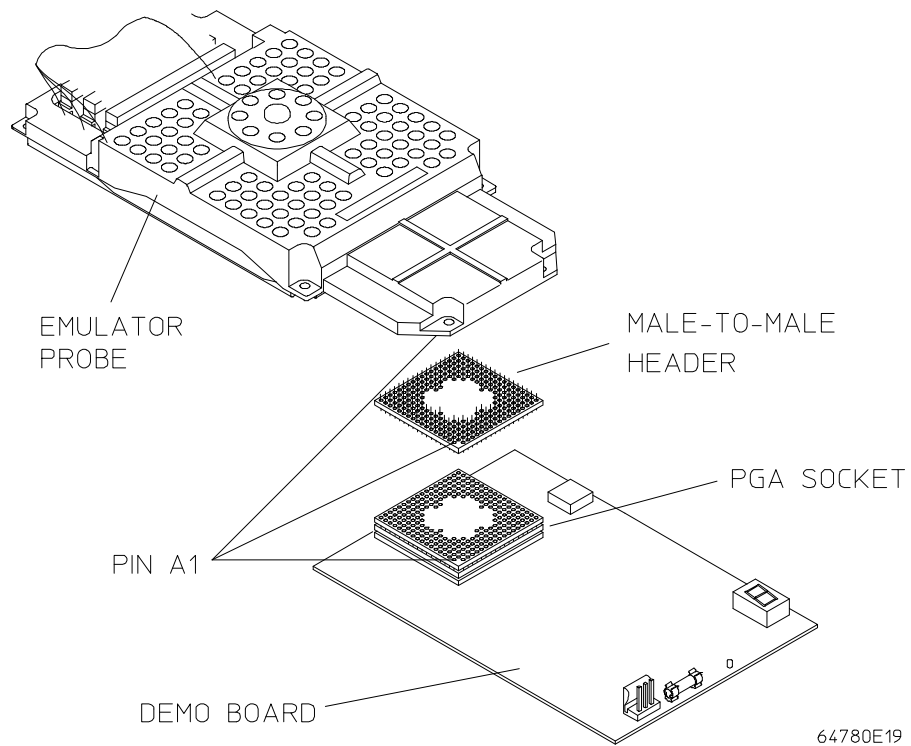
Adapters are available for special target system probing needs:

- Pin protectors, HP Part Number 5181-0206.

- PGA to PGA Flexible Adapter, Model Number HP E3430A.

- QFP Adapter.

These accessories have an electrical impact on your target system. In addition to delays, the accessories can cause problems with signal quality. Only use these accessories as a last resort.

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 5. Turn ON power

1 Turn ON power to the emulator.

2 Turn ON power to the target system.

If the emulator doesn't work with the target oscillator

Typically, an oscillator in the target system can be made to work with the emulator by adjusting the signal quality with a termination.

The Target Oscillator Clock Module, 64780-66509, connects the EXTAL line from the target system to the processor. It also provides a RC termination on this signal.

- If no termination or a different termination is required, a circuit can be constructed on a 14 pin header. Use the schematic diagram shown for the 64780-66509 as a guide. Pin 1 must be connected to pin 2 and pin 6 must be connected to pin 8.
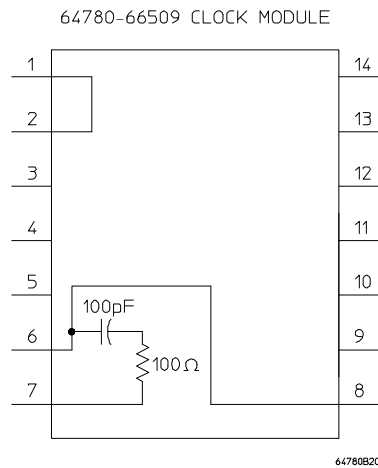
- If the EXTAL signal is not used on the target system, an internal oscillator can be used as an alternative. The clock module socket is arranged so that a standard 14 pin DIP TTL oscillator can be used. Target system power is provided on pin 14, ground on pin 7. The EXTAL connection to the processor is on pin 8. Pin 1 is pulled low so that oscillators with output enable functions can be used.

If the emulator doesn't work with the target crystal

Making a crystal circuit in the target system work with the emulator may be a problem, especially when an accessory such as a flexible cable is used to connect the emulator to the target system.

Target Crystal Clock Module B of the Clock Module Kit, 64780-66507, simply connects the XTAL and EXTAL signals to the processor and provides a typical value for the XFC bypass capacitance.

If the target circuit does not oscillate, an internal crystal circuit needs to be used. If the PLL has problems maintaining a lock, either the XFC capacitance needs to be adjusted, or there is too much noise on the clock.

- **If the circuit will not oscillate or there is too much noise on the clock, an internal crystal circuit should be used.**

If you are using either of the recommended standard crystal frequencies, place the appropriate clock module in the clock module socket. Use Clock Module E for 32.768 KHz operation, or use Clock Module C for 4.194 MHz operation. Both these modules provide the correct crystal circuit for use with the 68360 and a typical value for the XFC bypass capacitance.

If a different frequency is required, User Definable Clock Module D provides a platform to customize a clock circuit similar to the standard frequency clock modules.
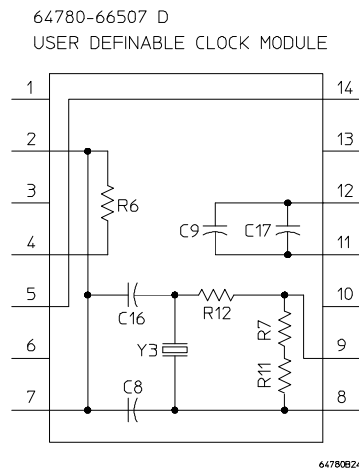
64780-66507 D
USER DEFINABLE CLOCK MODULE

Appropriate values should be chosen for the given topology. A parallel resonant crystal should be used. If the prescaler will be used, R6 should be loaded with a 0 Ohm resistor; otherwise, it should be left unloaded.

Alternatively, a crystal circuit can be implemented on a 14 pin header. Make sure all connections shown on the User Definable Clock Module are made on your header.

- If only the XFC bypass capacitance needs to be adjusted, a circuit can be constructed on a 14 pin header. Use the schematic diagram shown for Clock Module B as a guide. All the connections shown must be made, the only difference should be the capacitance value chosen.

```
64780-66507 B
TARGET CRYSTAL CLOCK MODULE
```

5
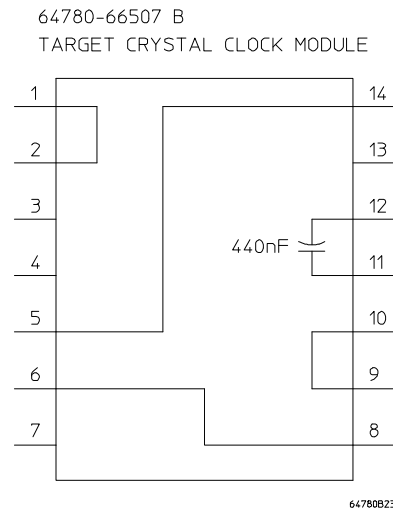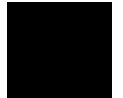
Configuring the Emulator

Configuring the Emulator
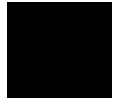
This chapter contains information about configuring the emulator.

- Setting the Hardware Options for an Emulator

- Setting the Hardware Options for an HP E3490A Software Probe

- Mapping Memory (Emulator Only)

- Selecting the Type of Monitor (Emulator Only)

- Using the EMSIM Registers

- Verifying the Emulator Configuration

- Setting Up the BNC Port (Emulator Only)

- Saving and Loading Configurations

- Setting the Real-Time Options

Setting the Hardware Options for an Emulator

This section shows you how:

- To select buffering for AS, DS, and R/W signals

- To enable or disable breaks on memory usage violations

- To enable or disable BERR to/from the target system

- To enable or disable external DMA

- To enable or disable target system interrupts

- To enable or disable break on writes to ROM

- To specify the external data bus size

- To specify the global chip select memory size after reset

- To specify the maximum bus speed

- To specify the emulation memory speed

- To select whether CLKO1 is driven to the target system

To select buffering for AS, DS, and R/W signals

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

2 Select Buffered or Unbuffered for the "Buffering of AS, DS, RW" option.

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

Buffering these signals improves signal quality and timing (see the emulator timing specifications that come with the emulator).

To enable or disable breaks on memory usage violations

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

2 Select Enable or Disable for the "Break on memory usage violation" option.

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

When enabled, any DRAM/Parity access into emulation memory causes a break.

When disabled, these accesses do not cause breaks.

To enable or disable BERR to/from the target system

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

2 Select Enable or Disable for the "BERR to/from target" option.

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

When enabled, the connection is complete for driving internal M68360 BERR signals to the target, and for delivering target BERR activity to the M68360.

When disabled, internal M68360 BERR signals will not be driven to the target, and the M68360 will not respond to target BERR activity.
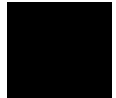
To enable or disable external DMA

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

2 Select Enable or Disable for the "External DMA" option.

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

When enabled, the emulator supports external DMA to the M68360 and supports DMA tracing, but does not support DMA accesses to emulation memory.

When disabled, the emulator does not support any external DMA activity.

These options do not affect internal DMA.

To enable or disable target system interrupts

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

2 Select Enable or Disable for the "Target System Interrupts" option.

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

When enabled, the emulator responds to interrupts generated by the target system while running in the user program or foreground monitor. All interrupts, including level 7 NMI, are blocked when execution is within the background monitor.

When disabled, the emulator ignores all interrupts generated by the target system, including level 7 NMI.

To enable or disable break on writes to ROM

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

2 Select Enable or Disable for the "Break on write to ROM" option.

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

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

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

To specify the external data bus size

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

2 Select 32 or 16 for the "External data bus size" option.

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

Specifies the width in bits.

To specify the global chip select memory size after reset

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

2 Select 32, 16, or 8 for the "Memory size accessed by global chip select when released from reset" option.

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

Specifies the width in bits.

To specify the maximum bus speed

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

2 Select 10 MHz, 25 MHz, 33.33 MHz, or 40 MHz for the "Maximum Bus Speed" option.

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

When 10MHz, the emulator does not add a wait state. You may program chip selects of the M68360 for any TCYC.

When 25MHz, the emulator does not add a wait state. Chip selects of the M68360 must be programmed for TCYC >= 1.

When 33.33MHz: When the memory type is HP 64172, the emulator will not add a wait state. Chip selects of the M68360 must be programmed for any TCYC >= 1. When the memory type is HP 64173 or both HP 64172 and HP 64173, the emulator will add one wait state. You may program the chip selects of the M68360 for TCYC >= 2.

When 40MHz, the emulator adds one wait state. Chip selects of the M68360 must be programmed for TCYC >= 2.

To specify the emulation memory speed

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

2 Select 64172/20ns, 64173/25ns, or Both for the "Memory Speed" option.

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

Identifies the type of memory modules installed on the emulator probe. This value is used in the wait state calculation associated with Maximum Bus Speed.

To select whether CLKO1 is driven to the target system

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

2 Select Buffered, Off, or Unbuffered for the "Clock O1 Mode" option.

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

Buffering this output signal improves signal quality.

Off specifies that CLKO1 is not driven to the target system.

Setting the Hardware Options for an HP E3490A Software Probe

The HP E3490A Software Probe supports only those features that are common on all the processors in the CPU32 family. Additional processor specific features and capabilities are supported through the use of user configurable parameters and through different interfaces. The parameters may be customized to suit specific target systems and saved in configuration files for future use.

This section shows you how:

- To set the processor type

- To enable or disable breaks on memory usage violations

After you have configured these values and you have set the initial values for the EMSIM registers, you need to "apply" the configuration by selecting Execution→Reset then Execution→Break. When the break occurs the following "configuration process" occurs:

- The target processor enters the BDM monitor.

- The BDM communication speed is set to the default clock rate.

- The emulation copies of the SIM registers are loaded into the target SIM.

- The BDM communication speed is set to the rate based on the value configured as the target processor clock rate.

To specify the processor type

In this interface, the processor type is automatically set to 68360. This configuration item is included for other Real-Time C Debugger products which support multiple processors.

To specify the target processor clock speed

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

2 Select a Processor Clock Rate from the list of options.

3 Click on the OK button.

The HP E3490A Software Probe needs to be configured to communicate at a rate which is compatible with your target processor clock rate.

If your target processor runs at less than 8 MHz after power-up, you will also need to set a configuration switch on the HP E3490A Software Probe.

The processor clock rate configuration parameter is used to specify the maximum rate that the HP E3490A Software Probe can communicate with the target processor through the BDM port. This maximum communication rate is based upon the target processor type and the target processor clock rate. The maximum communication rate directly impacts the performance of downloading files through the HP E3490A Software Probe and the intrusion time when polling. Running at a rate slower than the maximum does not prevent correct operation.

The default value for the processor clock rate configuration parameter is determined by the setting of configuration switch S2. In the Normal (CLOSED) position, this parameter is set to "greater than or equal to 8 MHz". In the OPEN position, this parameter is set to "greater than or equal to 131 kHz".

Most target systems run with a Processor Clock Rate of 8 MHz or above and so will function properly using the default value. For correct operation in target systems that run slower than 8 MHz, the default clock rate must be set to "greater than or equal to 131 kHz" by setting switch S2 to OPEN.

The best performance for the HP E3490A Software Probe usually requires changing the communication rate from the default value. Set the "Processor Clock Rate" parameter to the highest rate that is equal to or less than the target running clock rate. If using the internal clock synthesizer, set the EMSYNCR (EMulator copy of the SYNthesizer Control Register, SYNCR) to the same value as set by the initialization code.

See Also

"Using the EMSIM Registers"

"Detailed information about processor clock rates" in the *HP E3490A Software Probe User's Guide*.
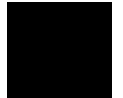
Mapping Memory (Emulator Only)

This section shows you how:

- To map memory

By default, the emulator assumes all memory accesses are made to RAM hardware in the target system. If you wish to use emulation memory hardware, or identify some of your target memory space as ROM hardware, you must enter these specifications in the memory map.

To map memory

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

2 Specify the starting address in the Start text box.

3 Specify the end address in the End text box.

4 If necessary, select the function code from the Func Code drop-down list.

5 Select the memory type in the Type option box.

6 If you are mapping a range of emulation memory, select the appropriate /DSACK Attributes option.

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

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).

The emulator provides two slots for emulation memory modules. The amount of emulation memory that can be mapped depends on the number, and size, of memory modules installed.

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

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

eram   Specifies "emulation RAM".

erom   Specifies "emulation ROM".

tram   Specifies "target RAM".

trom   Specifies "target ROM".

guarded   Specifies "guarded memory".

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

The /DSACK Attributes options specify where /DSACK signals come from for accesses in the range.

Processor = /DSACKs are from the processor.
Target = /DSACKs are from the target system.
default = 32-bit /DSACKs are from the emulator.
32 = 32-bit /DSACKs are from the emulator.
16 = 16-bit /DSACKs are from the emulator.
8 = 8-bit /DSACKs are from the emulator.

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

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

Example

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

```
┌─Define Map Term──────────────────────────┐
│ Start:      ┌─────────────┐   ┌───────────┐│
│             │ 6000        │   │  Apply    ││
│             └─────────────┘   └───────────┘│
│ End:        ┌─────────────┐                │
│             │ 0ffff       │                │
│             └─────────────┘                │
│ Func Code:  ┌──────────────────┐│          │
│             │ X              ▼ ││          │
│             └──────────────────┘│          │
│  ┌─Type──────────────────────────────────┐│
│  │  ◉ eram    ○ erom    ○ guarded        ││
│  │  ○ tram    ○ trom                     ││
│  └───────────────────────────────────────┘│
│  ┌─/DSACK Attributes─────────────────────┐│
│  │  ○ Processor ○ Target   ◉ default     ││
│  │  ○ 32         ○ 16       ○ 8          ││
│  └───────────────────────────────────────┘│
└──────────────────────────────────────────┘
```

Choose the Apply button to register the current map term.

Then, choose the Close button to quit mapping.

Selecting the Type of Monitor (Emulator Only)

This section shows you how:

- To select the background monitor

- To select the foreground monitor

- To use a custom foreground monitor

The monitor is the interface between the emulation system controller (which accepts and executes emulation commands) and the target system. The monitor uses the emulation microprocessor because that's the only way to access registers and target system memory.

When the emulation system controller recognizes that a command requires the monitor, it writes a command code to a communications area and "breaks" emulator execution into the monitor. The monitor reads this command (and any associated parameters), makes the appropriate accesses, places the values in the communication area, and returns emulator execution to its previous state.

Background Monitor

When a background monitor is selected, the Background Debug Mode (BDM) of the 68360 processor is used. The BKPT line is asserted to enter the monitor.

Interrupts from the target system are disabled during background monitor execution. If your programs have strict real-time requirements for servicing target system interrupts, you must use a foreground monitor program.

Foreground Monitor

The foreground monitor is an assembly language program that is executed by the 68360 emulation microprocessor in its normal operating mode.

When a foreground monitor is selected, the foreground monitor or downloaded custom monitor is loaded into emulation memory and consumes a 4-Kbyte block of the 68360's address range.

A foreground monitor has the following advantages and disadvantages:

Advantages

- The foreground monitor executes as a part of the user program, and target system interrupts can be enabled during monitor program execution for applications that have strict real-time processing requirements.

- The foreground monitor can be customized.

Disadvantages

- The foreground monitor occupies processor memory space.
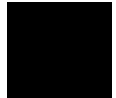
To select the background monitor

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

2 Select the Background option.

3 Choose the OK button.

When Background is selected, the Background Debug Mode (BDM) of the 68360 processor is used. The BKPT line is asserted to enter the monitor.

During background monitor operation, there will be no bus cycle activity except for memory reads and writes that result from memory display or modify commands.

Target system interrupts are blocked during background monitor operation.

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 4-Kbyte boundary (in other words, an address ending in 000H) and must be specified in hexadecimal. Four Kbytes of emulation memory must be mapped at this address.

4 If you want the foreground monitor to run at a lowered interrupt priority level in order to allow critical target system interrupts to be processed, select the desired interrupt priority level. When it's safe to lower the interrupt level, the foreground monitor will set the interrupt priority mask to either the level entered or the level in effect before monitor entry, whichever is greater.

5 Choose the OK button.

The foreground monitor, resident in emulator firmware, is automatically loaded into the specified 4-Kbyte block of emulation memory every time the emulator breaks into the monitor state from the reset state.

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 User Foreground option.

5 Enter the base address of the foreground monitor in the Monitor Address text box. The address must reside on a 4-Kbyte boundary (in other words, an address ending in 000H) and must be specified in hexadecimal. Four Kbytes of emulation memory must be mapped at this address.

6 If you want the foreground monitor to run at a lowered interrupt priority level in order to allow critical target system interrupts to be processed, select the desired interrupt priority level. When it's safe to lower the interrupt level, the foreground monitor will set the interrupt priority mask to either the level entered or the level in effect before monitor entry, whichever is greater.

7 Enter the name of the foreground monitor object file in the Monitor File Name text box.

8 Choose the OK button.

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\M360\FGMON directory. The file is named FGMON.S.

The custom foreground monitor is saved in the emulator (until the monitor type is changed) and reloaded into the specified 4-Kbyte block of emulation

memory every time the emulator breaks into the monitor state from the reset state.

Using the EMSIM Registers

This section shows you how:

- To view the SIM register differences

- To synchronize to the 68360 SIM registers

- To synchronize to the EMSIM registers

- To reset the EMSIM registers to processor defaults

The EMSIM registers behave differently, depending on whether you are using an emulator or an HP E3490A Software Probe.

EMSIM Registers in the Emulator

The 68360 processor contains a System Integration Module (SIM) which has the external bus interface, eight chip selects, and other circuitry to reduce external logic in a typical microprocessor system. The SIM can be programmed or configured in a variety of ways to suit the need of various systems.

The 68360 emulator contains circuitry that accommodates the flexibility of the SIM and maintains consistent emulation features.

The 68360 SIM is configured through the registers in the SIM register class; these registers control how the 68360 uses external signal lines to access memory.

The emulator is configured through the registers in the EMSIM register class. These registers control how the emulator interprets the signals from the 68360 when accessing emulation memory and passing information to the analysis trace.

Normally, the SIM and EMSIM registers should be programmed with the same values so they will be working together.

One of the primary functions of the EMSIM registers and associated logic is to provide A31-A28 to the memory mapper and analyzer so they will have the complete 32-bit address bus. Sending address lines to the mapper is easy if Port E of the 68360 is programmed as address lines; however, if Port E is programmed as write enables, the upper address lines are not available
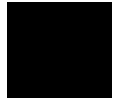
external to the 68360 (this is the case following reset). The chip selects, however, have access to the full 32 bit address inside the 68360. You can therefore locate memory using a chip select at an address that is not possible to decode externally. The emulator can use information in the programming of the chip selects to re-create the upper address lines. This provides a correct address in the analysis trace so that symbolic debugging is possible. Unforunately, these addresses cannot be re-created in time for memory mapping, so the mapper is limited to 28 bits of addressing when the upper address lines are not available.

Normally, the emulator is programmed through the EMSIM registers to match the programming of the 68360 SIM as it will exist after all of the boot-up configuration is complete. Programming of the emulator can be done before the boot-up code is run. In fact, the programming of the EMSIM registers is part of the configuration and will be loaded along with the memory map and other configuration items when a configuration file is loaded.

The default programming of the EMSIM registers matches the reset values of the 68360 SIM (refer to the Motorola *MC68360 User's Manual* for specific values).

Note that the emulator is programmed solely from the EMSIM register set and is therefore static with respect to the application program. No attempt is made to update the programming of the emulator by tracking instructions that will program the 68360 SIM.

EMSIM Registers in the HP E3490A Software Probe

Internal Registers

The CPU32 family of processors provides a variety of internal peripheral and memory modules that are directly connected to the CPU32 core through an internal bus. These modules are configured through memory mapped register banks. The base address of the register banks as well as the base address of internal memory modules are established through Module Configuration Registers (e.g. MCR) and Base Address Registers (e.g. RAMBAR or MBAR). A common module throughout the family is a System Integration Module (SIM) which controls such things as clock speed and external chip selects.

How Internal Register Values are Set

These registers are typically initialized by the CPU32 executing the reset initialization code. During development this code may not be available or may not exist on the target system. To aid in development, the most important of these registers can be set directly by the HP E3490A Software Probe. This enables such functions as clock speed, chip selects, and location of internal memory to be established prior to executing any user code. Once these registers are set, resources in the target system can be accessed in the same manner as the processor would access them after executing the reset initialization code. Activities such as downloading code into the target system can now be performed through the HP<N>E3490A Software Probe.

The emulator copy is identified by the prefix "EM" on the register name (e.g. EMSYNCR is the emulator copy of the SYNCR register) and are referred to as the EMSIM. The EMSIM registers are transferred to the processor registers when the target processor is reset while it is running in the BDM monitor.

The names and values of the EMSIM registers are displayed in the Emulator SIM Registers window.

Based on the previous discussion, it should be clear that the EMSIM values specified during configuration need to match the intended programming and use of your CPU32 target system. You need to carefully decide how the processor will be configured and the corresponding SIM values.

Methods for Configuring EMSIM Register Values

There are two methods you can use to configure EMSIM register values:

- Copy values from the SIMs into the EMSIM registers.

- Manually define each of the EMSIM values using the debugger interface.

Note   The HP E3490A Software Probe supports configuration of the internal registers in the System Integration Module (SIM) and other important Module Configuration Registers and Base Address Registers. To simplify the interface, all configurable registers will be referred to as SIM registers even if they are technically part of another module.

Note   Some registers can only be written once after processor reset.

121

If you set the EMSIM values, then reset and break, the EMSIM values will be written to the SIM registers. If your initialization code then attempts to write to one of the "write once after reset" registers, the writes will fail. In this case, you must use the Execution→Run...From Reset command to correctly execute the initialization code.

See Also

"Internal Representation of SIM and EMSIM Registers" in the *HP E3490A Software Probe User's Guide.*

To view the SIM register differences

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Show differences for M68360 and emsim registers" from the Synchronize SIM registers list.

3 Choose the Apply/Results button to display the differences in the viewing area.

Note

(For HP E3490A Software Probe only.) The HP E3490A Software Probe, when comparing SIM and EMSIM registers, will not compare EMSIM registers that have not been set. When you first turn on the HP E3490A Software Probe and the target system and start the Real-Time C Debugger, the Show differences for processor and emsim regs command will not find any differences. If one register is modified, just that one register may show differences. A complete check of the register differences will occur only if a complete configuration is loaded or the SIM registers are copied to the EMSIM registers. Once written, any EMSIM register will be considered valid until you cycle power on the HP E3490A Software Probe.

To synchronize to the 68360 SIM registers

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I)
command.

2 Select "Synchronize from '360 sim regs, copy to emsim regs" from the
Synchronize SIM registers list.

3 Choose the Apply/Results button.

This is useful if initialization code that configures the 68360 SIM exists, but
you don't know what its values are. In this case, you can use the default
configuration, run from reset to execute the initialization code, and
synchronize the EMSIM registers to match the 68360 SIM.

To synchronize to the EMSIM registers

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I)
command.

2 Select "Synchronize from emsim regs, copy to '360 registers" from the
Synchronize SIM registers list.

3 Choose the Apply/Results button.

Synchronization to the EMSIM registers happens automatically each time a
break to the monitor from emulation reset occurs; this ensures that the 68360
is prepared to properly access memory when a program is downloaded to the
emulator.

Note   (For HP E3490A Software Probe only.) Some registers can only be written once after processor reset. If you set the EMSIM values, then reset and break, the EMSIM values will be written to the SIM registers. If your initialization code then attempts to write to one of the "write once after reset" registers, the writes will fail. In this case, you must use the Execution→Run...From Reset command to correctly execute the initialization code.

To reset the EMSIM registers to processor defaults

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Default the emsim register set" from the Synchronize SIM registers list.

3 Choose the Apply/Results button.

This resets the EMSIM registers to the processor's default (power-up) values.

Verifying the Emulator Configuration

This section shows you how:

- To check for configuration inconsistencies

- To display information about chip selects

- To display information about bus interface ports

- To display information about the memory map

- To display information about the reset mode configuration

- To display information about the upper address mode

- To display information about the clock input mode

- To display assembly code for setting up the SIM

To check for configuration inconsistencies

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Check emulator configuration" from the Config and SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the viewing area.

This command:

- Checks for inconsistencies between the reset mode configuration value and the EMSIM registers.

- Compares corresponding values in the SIM and EMSIM register sets.

To display information about chip selects

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I)
command.

2 Select "Chip selects in SIM (processor) register set" or "Chip selects
in EMSIM (emulator) register set" from the Config and SIM
Programming Info. list.

3 Choose the Display Info. button to display the information in the
viewing area.

The resulting display shows how the chip select is assigned, the base address,
and other information from the option register.

To display information about bus interface ports

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I)
command.

2 Select "Bus interface ports in SIM (processor) register set" or "Bus
interface ports in EMSIM (emulator) register set" from the Config and
SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the
viewing area.

The resulting display shows the pin assignments for Port E.

To display information about the memory map

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Memory map & correlation with CSs, IM reg blk & RAM" from the Config and SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the viewing area.

The resulting display shows detailed information about the memory map.

To display information about the reset mode configuration

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Reset mode configuration value and operation" from the Config and SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the viewing area.

The resulting display shows the data bus size and global chip select memory access size.

To display information about the upper address mode

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Upper address mode of emulator for 68360" from the Config and SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the viewing area.

The resulting display shows whether the upper address lines are used as A31-A28 or WE3-WE0.

To display information about the clock input mode

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Clock input mode" from the Config and SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the viewing area.

The clock input mode depends on the clock module installed in the emulation probe. For details, refer to the documentation that comes with the 68360 emulation probe.

To display assembly code for setting up the SIM

1 Choose the Settings→Emulator Config→Information... (ALT, S, E, I) command.

2 Select "Assembly listing matching current EMSIM registers" from the Config and SIM Programming Info. list.

3 Choose the Display Info. button to display the information in the viewing area.

The resulting display shows the assembly language program that will initialize the processor as defined by the current EMSIM register contents.
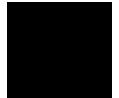
Setting Up the BNC Port (Emulator Only)

This section shows you how:

- To output the trigger signal on the BNC port

- To receive an arm condition on the BNC port

To output the trigger signal on the BNC port

- Choose the Settings→BNC→Outputs Analyzer Trigger (ALT, S, B, 0) 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.

See Also

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

To load an emulator configuration

1 Choose the File→Load Emulator Config... (ALT, F, E) command.

2 Select the name of the emulator configuration command file to load from the file selection dialog box.

3 Choose the OK button.

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

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

Setting the Real-Time Options

This section shows you how:

- To allow or deny monitor intrusion

- To turn polling ON or OFF

The monitor program is executed by the emulation microprocessor when target system memory, memory-mapped I/O, and microprocessor registers are displayed or edited. Also, periodic polling to update the Memory, I/O, WatchPoint, and Register windows can cause monitor program execution.
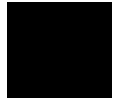
This means that when the user program is running and monitor intrusion is allowed, the user program must be temporarily interrupted in order to display or edit target system memory, display or edit registers, or update window contents.

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

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

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

To allow or deny monitor intrusion

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

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

When you deny monitor intrusion, any debugger command that may interrupt a running user program is prevented. This ensures the user program will 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.

6

Debugging Programs

Debugging Programs
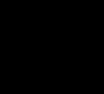
This chapter contains information on loading and debugging programs.

- 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 (Emulator Only)

- Setting Up Custom Trace Specifications

- Programming Target Flash Memory (HP E3490A Software Probe Only)

If you encounter problems when using the emulator/analyzer, refer to the chapter titled "Solving Problems" in the MC68360 Emulator/Analyzer Installation/Service/Terminal Interface Manual.

Loading and Displaying Programs

This section shows you how:

- To load user programs

- To display source code only

- To display source code mixed with assembly instructions

- To display source files by their names

- To specify source file directories

- To search for function names in the source files

- To search for addresses in the source files

- To search for strings in the source files

To load user programs

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

2 Select the function code of the memory space into which the program should be loaded.

3 Select the file to be loaded.

4 Choose the Load button to load the program.

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

With this command, you can load any IEEE-695 object file created with any of the Microtec or HP programming tools for 68360 (CPU32).

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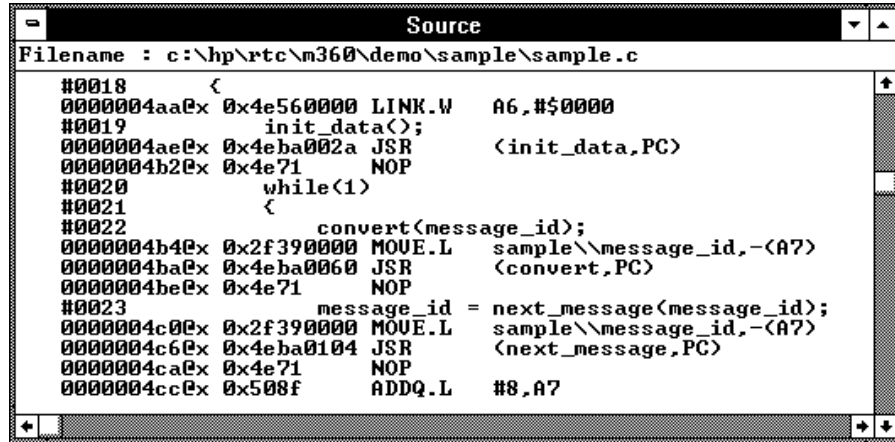
The source only display shows line numbers with the source code.

To display source code mixed with assembly instructions

1 Position the cursor on the starting line to be displayed.

2 From the Source window or Trace 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

```
┌─────────────────────────────────────────────────────────────────────┐
│ ▭                          Source                               ▼ ▲  │
├─────────────────────────────────────────────────────────────────────┤
│Filename : c:\hp\rtc\m360\demo\sample\sample.c                        │
├─────────────────────────────────────────────────────────────────────┤
│    #0018        {                                                 ✦  │
│    0000004aa@x 0x4e560000 LINK.W    A6,#$0000                         │
│    #0019         init_data();                                        │
│    0000004ae@x 0x4eba002a JSR       <init_data,PC>                    │
│    0000004b2@x 0x4e71      NOP                                        │
│    #0020          while(1)                                           │
│    #0021          {                                                  │
│    #0022            convert(message_id);                             │
│    0000004b4@x 0x2f390000 MOVE.L    sample\\message_id,-(A7)          │
│    0000004ba@x 0x4eba0060 JSR       <convert,PC>                      │
│    0000004be@x 0x4e71      NOP                                        │
│    #0023            message_id = next_message(message_id);           │
│    0000004c0@x 0x2f390000 MOVE.L    sample\\message_id,-(A7)          │
│    0000004c6@x 0x4eba0104 JSR       <next_message,PC>                 │
│    0000004ca@x 0x4e71      NOP                                        │
│    0000004cc@x 0x508f      ADDQ.L   #8,A7                             │
├─────────────────────────────────────────────────────────────────────┤
│ ← ▭                                                              → ▼  │
└─────────────────────────────────────────────────────────────────────┘
```

To display source files by their names

1 Make the Source window the active window, and choose the
Display→Select Source... (ALT, -, D, L) command from the Source
window's control menu.

2 Select the desired file.

3 Choose the Select button.

4 Choose the Close button.

Note   The contents of assembly language source files cannot be displayed.

To specify source file directories

1 Make the Source window the active window, and choose the Display→Select Source... (ALT, -, D, L) command from the Source window's control menu.

2 Choose the Directory... button.

3 Enter the directory name in the Directory text box.

4 Choose the Add button.

5 Choose the Close button to close the Search Directories dialog box.

6 Choose the Close button to close the Select Source dialog box.

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

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

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

To search for function names in the source files

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

2 Select the function to be searched.

3 Choose the Find button.

4 Choose the Close button.

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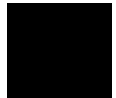
To search for addresses in the source files

1 From the Source window's control menu, choose the
Search→Address... (ALT, -, R, A) command.

2 Type or paste the address into the Address text box.

3 Choose the Find button.

4 Choose the Close button.

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

To search for strings in the source files

1 From the Source window's control menu, choose the
Search→String... (ALT, -, R, S) command.

2 Type or paste the string into the String text box.

3 Select whether the search should be case sensitive.

4 Select whether the search should be down (forward) or up
(backward).

5 Choose the Find Next button. Repeat this step to search for the next
occurrence of the string.

6 Choose the Cancel button to close the dialog box.

Displaying Symbol Information

This section shows you how:

- To display program module information

- To display function information

- To display external symbol information

- To display local symbol information

- To display global assembler symbol information

- To display local assembler symbol information

- To create a user-defined symbol

- To display user-defined symbol information

- To delete a user-defined symbol

- To display the symbols containing the specified string

To display program module information

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

To display function information

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

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

Example

Function Information Display
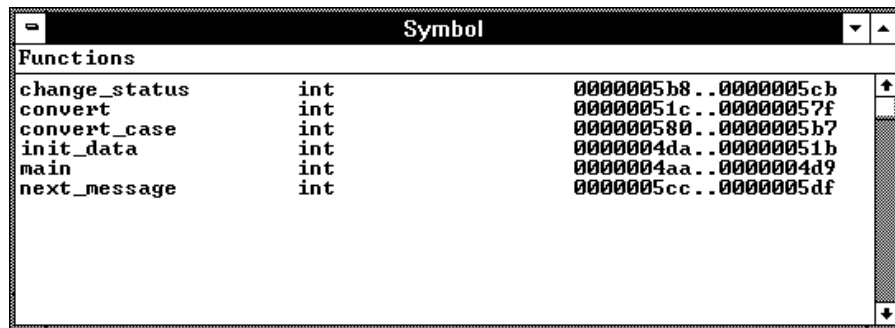
```
┌─┬────────────────────────────────────────────────────────┬─┬─┐
│ ─ │                    Symbol                            │▼│▲│
├───┴────────────────────────────────────────────────────────┴─┴─┤
│Functions                                                        │
├─────────────────────────────────────────────────────────────┬──┤
│change_status      int              0000005b8..0000005cb     │▲ │
│convert            int              00000051c..00000057f     │▓ │
│convert_case       int              000000580..0000005b7     │▓ │
│init_data          int              0000004da..00000051b     │▓ │
│main               int              0000004aa..0000004d9     │  │
│next_message       int              0000005cc..0000005df     │  │
│                                                             │  │
│                                                             │  │
│                                                             │  │
│                                                             │  │
│                                                             │▼ │
└─────────────────────────────────────────────────────────────┴──┘
```

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
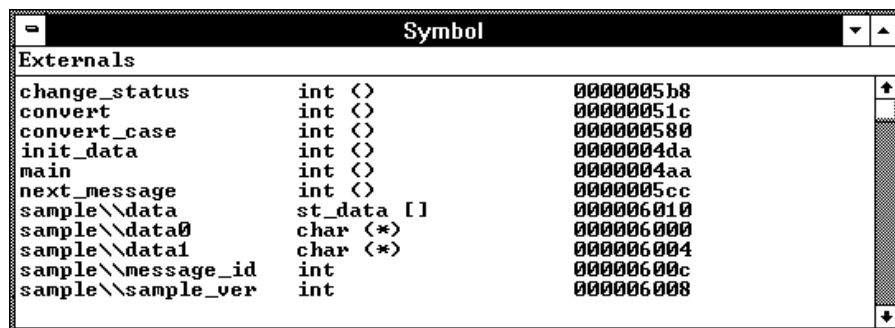
```
┌─┬─────────────────────────────── Symbol ──────────────────────────┬─┬─┐
│ ▪ │                                                                │ ▼ │ ▲ │
├───────────────────────────────────────────────────────────────────────┤
│ Externals                                                              │
├───────────────────────────────────────────────────────────────────────┤
│ change_status      int ()           0000005b8                    │ ▲ │
│ convert            int ()           00000051c                    │   │
│ convert_case       int ()           000000580                    │   │
│ init_data          int ()           0000004da                    │   │
│ main               int ()           0000004aa                    │   │
│ next_message       int ()           0000005cc                    │   │
│ sample\\data       st_data []        000006010                    │   │
│ sample\\data0      char (*)          000006000                    │   │
│ sample\\data1      char (*)          000006004                    │   │
│ sample\\message_id int              00000600c                    │   │
│ sample\\sample_ver int              000006008                    │   │
│                                                                  │ ▼ │
└───────────────────────────────────────────────────────────────────────┘
```
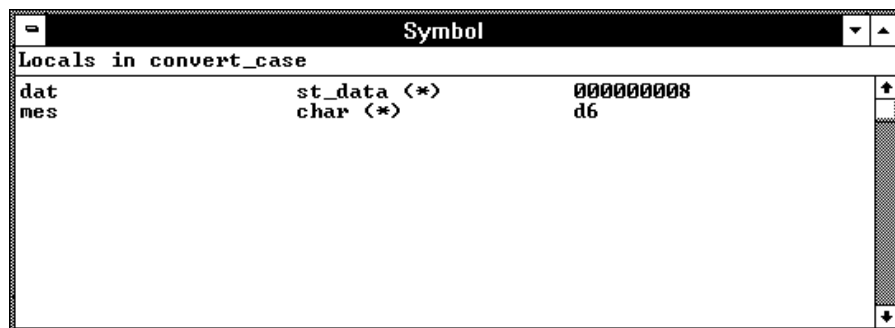
To display local symbol information

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

```
┌────────────────────────────────────────────────────────────────┐
│  ▬                         Symbol                        ▼ ▲    │
├────────────────────────────────────────────────────────────────┤
│ Locals in convert_case                                         │
├────────────────────────────────────────────────────────────────┤
│ dat                  st_data (*)        000000008          ▲   │
│ mes                  char (*)           d6                      │
│                                                                 │
│                                                                 │
│                                                                 │
│                                                                 │
│                                                                 │
│                                                            ▼   │
└────────────────────────────────────────────────────────────────┘
```
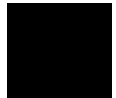
To display global assembler symbol information

• 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
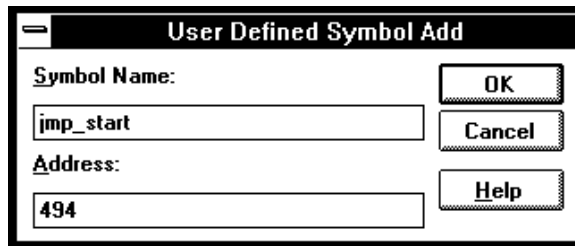module are displayed.

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":

```
┌─────────────────────────────────────────┐
│ ▬        User Defined Symbol Add         │
├─────────────────────────────────────────┤
│ Symbol Name:                  ┌────────┐ │
│                               │   OK   │ │
│ ┌─────────────────────────┐   └────────┘ │
│ │ jmp_start               │   ┌────────┐ │
│ └─────────────────────────┘   │ Cancel │ │
│ Address:                      └────────┘ │
│                               ┌────────┐ │
│ ┌─────────────────────────┐   │  Help  │ │
│ │ 494                     │   └────────┘ │
│ └─────────────────────────┘              │
└─────────────────────────────────────────┘
```
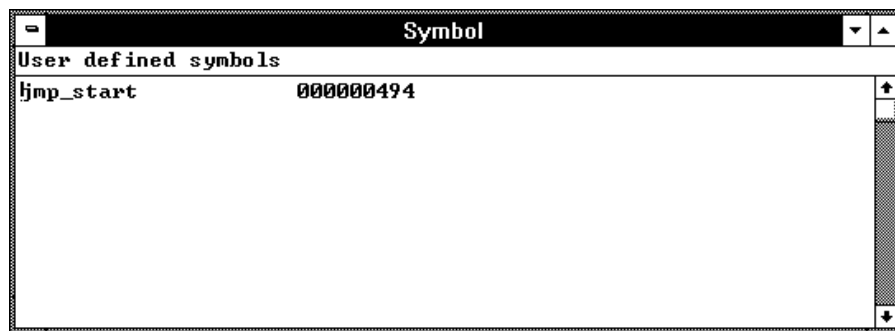
To display user-defined symbol information

- 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.
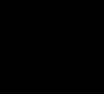
To display the symbols containing the specified string

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.

To step over a function

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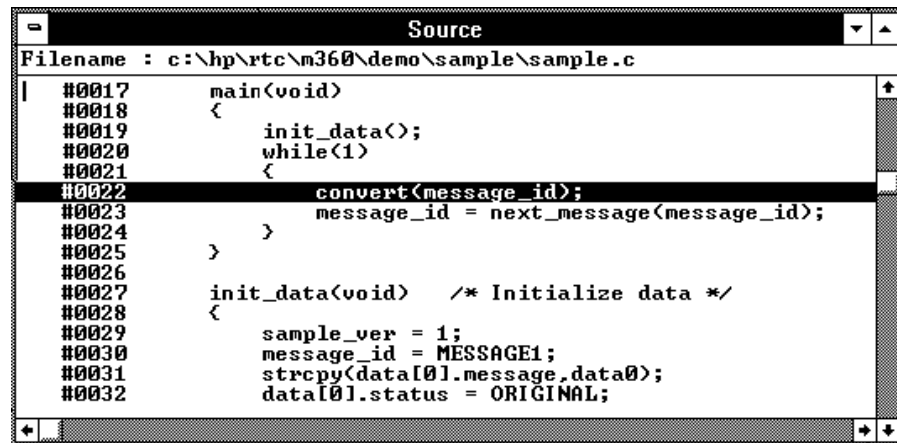
- Or, press the F3 key.

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

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

JSR

BSR

Example

```
 ▭                            Source                            ▼ ▲
Filename : c:\hp\rtc\m360\demo\sample\sample.c
│   #0017        main(void)                                         ⬧
    #0018        {
    #0019            init_data();
    #0020            while(1)
    #0021            {
    #0022                convert(message_id);
    #0023                message_id = next_message(message_id);
    #0024            }
    #0025        }
    #0026
    #0027        init_data(void)     /* Initialize data */
    #0028        {
    #0029            sample_ver = 1;
    #0030            message_id = MESSAGE1;
    #0031            strcpy(data[0].message,data0);
    #0032            data[0].status = ORIGINAL;
 ◄ ▯                                                              ► ↓
```

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

To step multiple lines or instructions

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

2 Select one of the Current PC, Start Address, or Address options.
(Enter the starting address when the Address option is selected.)

3 In the Count text box, type the number of lines to be single-stepped.

4 Choose the Execute button.

5 Choose the Close button to close the dialog box.

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

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

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

To step over functions, select the Over check box.
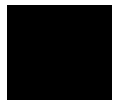
To run the program until the specified line

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

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

Execution stops immediately before the cursor-selected line.

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

If the specified address is not reached within the number of milliseconds specified by StepTimerLen in the B3627.INI file, a dialog box appears, asking you to cancel the command by choosing the Stop button. When the Stop button is chosen, the program execution stops, the breakpoint is deleted, and the processor transfers to the RUNNING IN USER PROGRAM status.

Note

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

To run the program until the current function return

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

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

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

To run the program from a specified address

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

2 Select one of the Current PC, Start Address, User Reset, or Address
options. (Enter the address when the Address option is selected.)

3 Choose the Run button.

The Current PC option executes the program from the current program
counter address.

The Start Address option executes the program from the *transfer address*.

The User Reset option resets the emulation processor and lets the emulator
run and fetch its stack pointer and program counter value from memory.

The Address option executes the program from the address specified.

To stop program execution

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

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

To reset the processor

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

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

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

Note that the emulator performs a reset by executing an external hard reset (RESETH). This causes the controller to assert the following reset lines: INTRST, INTSYSRST, CLKRST, and EXTSYSRST.

Using Breakpoints and Break Macros

This section shows you how:

- To set a breakpoint

- To disable a breakpoint

- To delete a single breakpoint

- To list the breakpoints and break macros

- To set a break macro

- To delete a single break macro

- To delete all breakpoints and break macros

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

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

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

All breakpoints are deleted when RTC is exited.
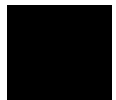
To set a breakpoint

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

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

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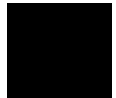
Example

To set a breakpoint at line 56:
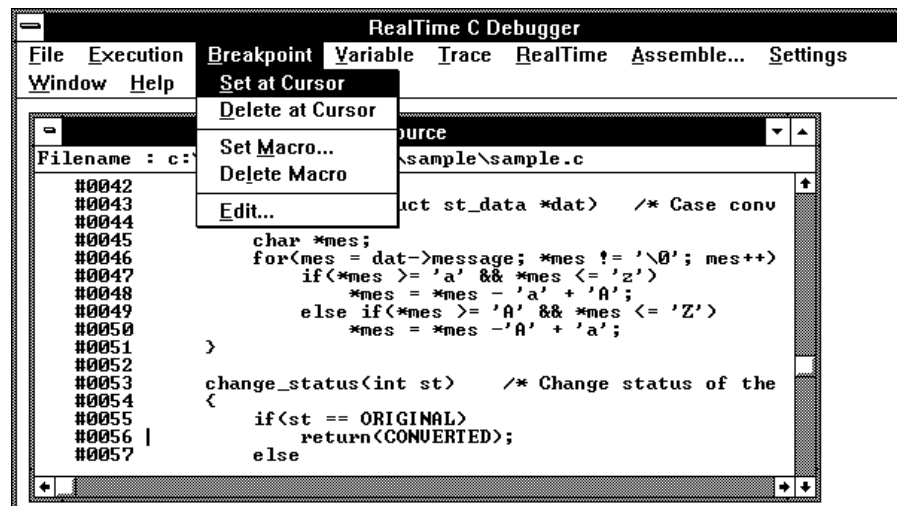
Note

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

To disable a breakpoint

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

2 Select the breakpoint to be disabled.

3 Select the Enable/Disable button. Notice that "DI" appears next to the breakpoint in the list.

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

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

To delete a single breakpoint

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

Or:

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

2 Select the breakpoint to be deleted.

3 Choose the Delete button.

4 Choose the Close button.

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

To list the breakpoints and break macros

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

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

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

To set a break macro

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

2 Choose the Breakpoint→Set Macro... (ALT, B, M) command.

3 Select the Add Macro check box in the Breakpoint Edit dialog box.

4 Specify the macro command in the Macro Command text box.

5 Choose the Set button.

6 To add another macro command, repeat steps 4 and 5.

7 To exit the Breakpoint Edit dialog box, choose the Close button.

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

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

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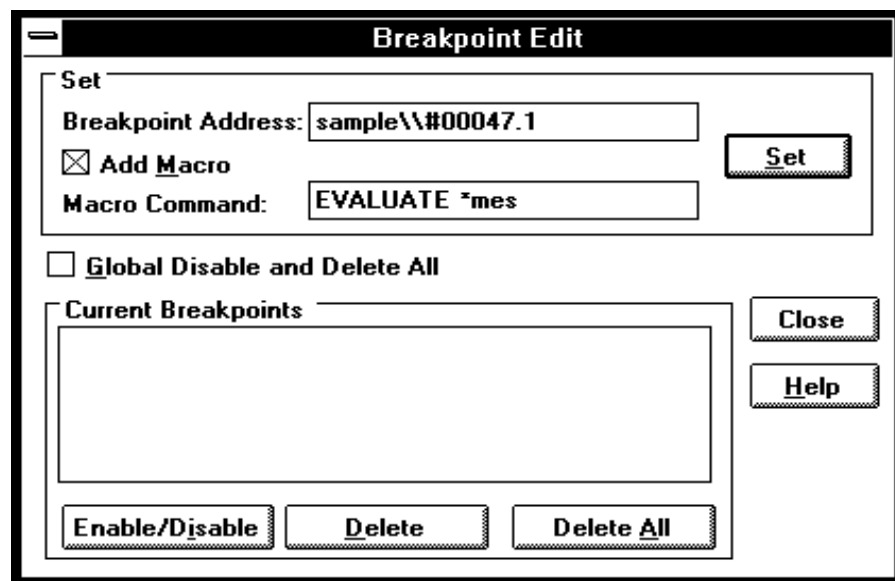
Example

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

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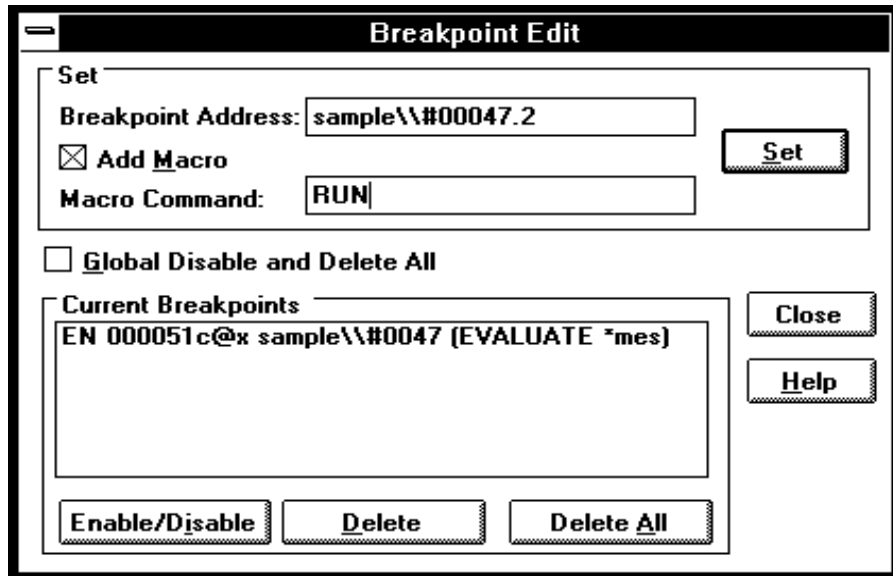
Select the Add Macro check box.

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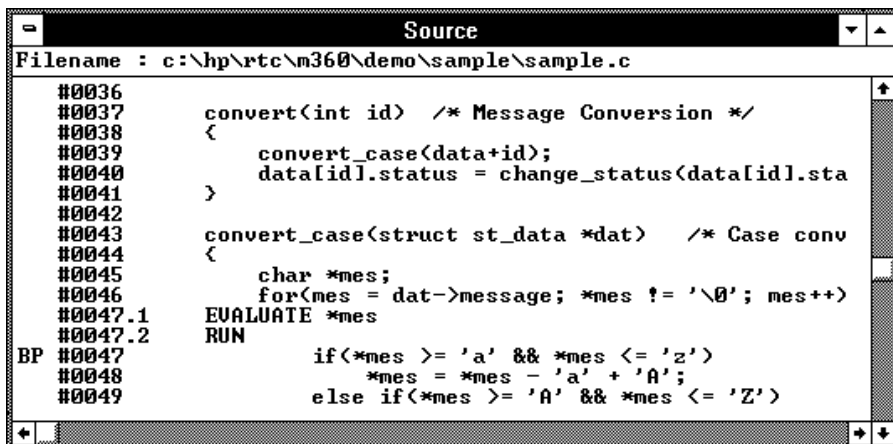

Choose the Set button.

Enter "RUN" in the Macro Command text box.

Choose the Set button.

Choose the Close button.

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

To delete a single break macro

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

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

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

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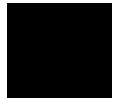
To delete all breakpoints and break macros

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

2 Choose the Delete All button.

3 Select the Global Disable and Delete All check box.

4 Choose the Close button.

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

Displaying and Editing Variables

This section shows you how:

- To display a variable

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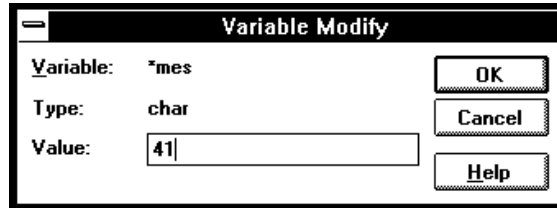
- To monitor a variable in the WatchPoint window

To display a variable

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

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

3 Choose the Update button to read the contents of the variable and display the value in the dialog box.

4 To exit the Variable dialog box, choose the Close button.

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

	Variable Modify	
Variable:	*mes	**OK**
Type:	char	**Cancel**
Value:	41	**Help**

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 of the function.
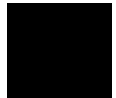
To monitor a variable in the WatchPoint window

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 of the function.

Displaying and Editing Memory

This section shows you how:

- To display memory

- To edit memory

- To copy memory to a different location

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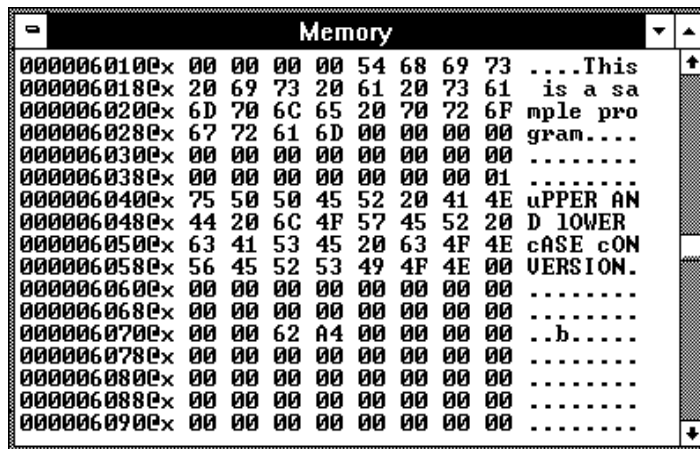
To change the size of the data displayed, access the Memory window's control menu; then, choose the Display→Byte (ALT, -, D, Y), Display→16 Bits (ALT, -, D, 1), or Display→32 Bits (ALT, -, D, 3) command. When the Display→Byte (ALT, -, D, Y) command is chosen, ASCII values are also displayed.

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

```
┌─────────────────────────────────────────────────┐▼▲
│ ▬              Memory                       ▼ ▲  │
├─────────────────────────────────────────────────┤
│ 000006010@x  00 00 00 00 54 68 69 73  ....This  │↑
│ 000006018@x  20 69 73 20 61 20 73 61   is a sa  │
│ 000006020@x  6D 70 6C 65 20 70 72 6F  mple pro  │
│ 000006028@x  67 72 61 6D 00 00 00 00  gram....  │
│ 000006030@x  00 00 00 00 00 00 00 00  ........  │
│ 000006038@x  00 00 00 00 00 00 00 01  ........  │
│ 000006040@x  75 50 50 45 52 20 41 4E  uPPER AN  │
│ 000006048@x  44 20 6C 4F 57 45 52 20  D lOWER   │
│ 000006050@x  63 41 53 45 20 63 4F 4E  cASE cON  │
│ 000006058@x  56 45 52 53 49 4F 4E 00  VERSION.  │
│ 000006060@x  00 00 00 00 00 00 00 00  ........  │
│ 000006068@x  00 00 00 00 00 00 00 00  ........  │
│ 000006070@x  00 00 62 A4 00 00 00 00  ..b.....  │
│ 000006078@x  00 00 00 00 00 00 00 00  ........  │
│ 000006080@x  00 00 00 00 00 00 00 00  ........  │
│ 000006088@x  00 00 00 00 00 00 00 00  ........  │
│ 000006090@x  00 00 00 00 00 00 00 00  ........  │↓
└─────────────────────────────────────────────────┘
```

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.
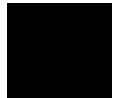
To copy memory to a different location

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 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.

2 From the Memory window's control menu, choose the Utilities→Store... (ALT, -, U, S) command.

3 Enter the starting address in the Start text box.

4 Enter the end address in the End text box.

5 Enter a file name in the File Name text box.

6 Choose the Export button.

7 Re-map the address range as emulation memory.

8 From the Memory window's control menu, choose the Utilities→Load... (ALT, -, U, L) command.

9 Enter the file name in the File Name text box.

10 Choose the Import button.

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

The following commands cannot be used when programs are stored in target system ROM. However, you can use these commands if you copy the contents of target system ROM into emulation memory:

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)

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 0ffffh, for the string "This":

```
┌─────────────────────────────────────────────┐
│ ▬           Search Memory                     │
├─────────────────────────────────────────────┤
│ Value:   ┌──────────────┐    ┌──────────┐    │
│          └──────────────┘    │  Search  │    │
│ String:  ┌──────────────┐    └──────────┘    │
│          │ This         │    ┌──────────┐    │
│          └──────────────┘    │  Close   │    │
│ Start:   ┌──────────────┐    └──────────┘    │
│          │ 6000         │    ┌──────────┐    │
│          └──────────────┘    │  Help    │    │
│ End:     ┌──────────────┐    └──────────┘    │
│          │ 0ffff        │                     │
│          └──────────────┘                     │
│                                               │
│ Size:    ◉ byte   ○ 16 Bits   ○ 32 Bits      │
└─────────────────────────────────────────────┘
```
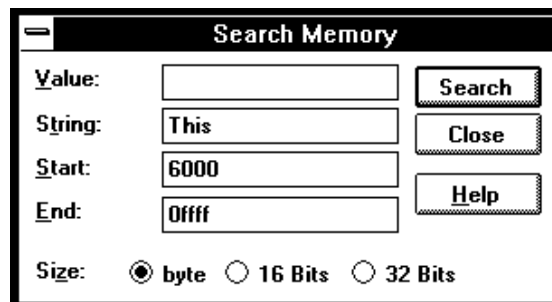
Displaying and Editing I/O Locations

This section shows you how:

- To display I/O locations

- To edit an I/O location

With the 68360 microprocessor, I/O locations are memory-mapped.

To display I/O locations

1 Choose the Window→I/O command.

2 From the I/O window's control menu, choose the Define... (ALT, -, D) command.

3 Enter the address in the Address text box.

4 Select whether the size of the I/O location is a Byte or 16 Bits.

5 Choose the Set button.

6 Choose the Close button.

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

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

Example   To display the contents of address 2000:

```
┌──────────────────────────────────────────────┐
│ ▬                   I/O Setting                │
├──────────────────────────────────────────────┤
│ ┌─Set──────────────────────────────────────┐ │
│ │  Address:    ┌────────────────┐           │ │
│ │              │ 2000           │           │ │
│ │              └────────────────┘           │ │
│ │  Size:      ◉ byte      ○ 16 Bits         │ │
│ │                                ┌────────┐ │ │
│ │  Space:     ◉ Mem              │  Set   │ │ │
│ │                                └────────┘ │ │
│ └──────────────────────────────────────────┘ │
│ ┌─I/O set──────────────────────┐             │
│ │ address: 000002000, size: byte│             │
│ │                               │  ┌────────┐ │
│ │                               │  │ Close  │ │
│ │                               │  └────────┘ │
│ │                               │             │
│ │                               │  ┌────────┐ │
│ │                               │  │ Help   │ │
│ │                               │  └────────┘ │
│ └───────────────────────────────┘            │
│ ┌────────┐                                    │
│ │ Delete │                                    │
│ └────────┘                                    │
└──────────────────────────────────────────────┘
```
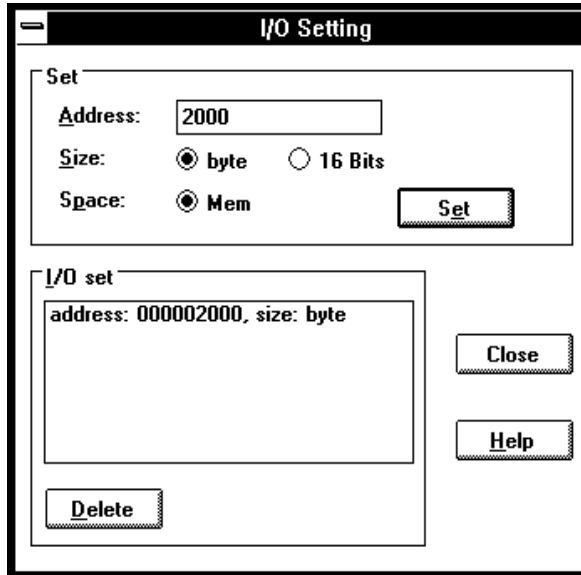
To edit an I/O location

1 Display the I/O value to be changed with the Window→I/O command.

2 Double-click the value to be changed.

3 Use the keyboard to enter a new value.

4 Press the Enter key.

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

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

Displaying and Editing Registers

This section shows you how:

- To display registers

- To edit registers
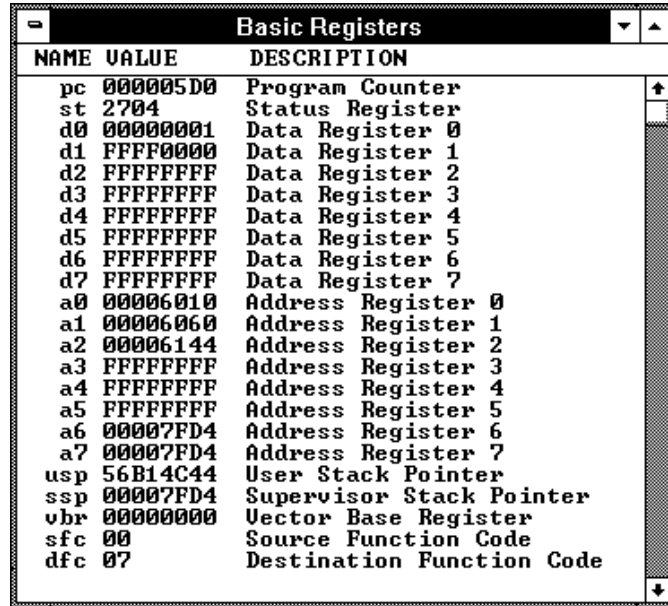
To display registers

- Choose the Window→Register command.

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

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

Example   Register Contents Displayed in the Register Window

```
┌─────────────────────────────────────────────┐
│ ▬           Basic Registers            ▼ ▲ │
├─────────────────────────────────────────────┤
│ NAME VALUE      DESCRIPTION                  │
│  pc 000005D0    Program Counter          ↑  │
│  st 2704        Status Register          ▓  │
│  d0 00000001    Data Register 0          ▓  │
│  d1 FFFF0000    Data Register 1             │
│  d2 FFFFFFFF    Data Register 2             │
│  d3 FFFFFFFF    Data Register 3             │
│  d4 FFFFFFFF    Data Register 4             │
│  d5 FFFFFFFF    Data Register 5             │
│  d6 FFFFFFFF    Data Register 6             │
│  d7 FFFFFFFF    Data Register 7             │
│  a0 00006010    Address Register 0          │
│  a1 00006060    Address Register 1          │
│  a2 00006144    Address Register 2          │
│  a3 FFFFFFFF    Address Register 3          │
│  a4 FFFFFFFF    Address Register 4          │
│  a5 FFFFFFFF    Address Register 5          │
│  a6 00007FD4    Address Register 6          │
│  a7 00007FD4    Address Register 7          │
│ usp 56B14C44    User Stack Pointer          │
│ ssp 00007FD4    Supervisor Stack Pointer    │
│ vbr 00000000    Vector Base Register        │
│ sfc 00          Source Function Code        │
│ dfc 07          Destination Function Code ↓ │
└─────────────────────────────────────────────┘
```

To edit registers

1 Display the register contents by choosing the Window→Register command.

2 Double-click the value to be changed.

3 Use the keyboard to enter a new value.

4 Press the Enter key.

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

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

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

Tracing Program Execution (Emulator Only)

This section shows you how:

- To trace function flow

- To trace callers of a specified function

- To trace execution within a specified function

- To trace accesses to a specified variable

- To trace before a particular variable value and break

- To trace until the command is halted

- To stop a running trace

- To repeat the last trace

- To display bus cycles

- To display absolute or relative counts

- To change the disassembly of bus cycle data

- To display dequeued trace data

- If you are having problems tracing

How the Analyzer Works

Trace data is available only if you are using an emulator. The HP E3490A Software Probe does not have a trace analyzer.

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

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

The store condition limits the kinds of states that are stored in trace memory.
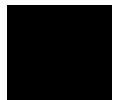
When the states stored are limited by the store condition, up to two states which satisfy the prestore condition may be stored when they occur before the states that satisfy the store condition.

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

Note   The analyzer traces unexecuted instructions due to prefetching done by the microprocessor.

Trace Window Contents

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

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

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

To trace function flow

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

The command stores function entry points, and the resulting trace shows
program execution flow.
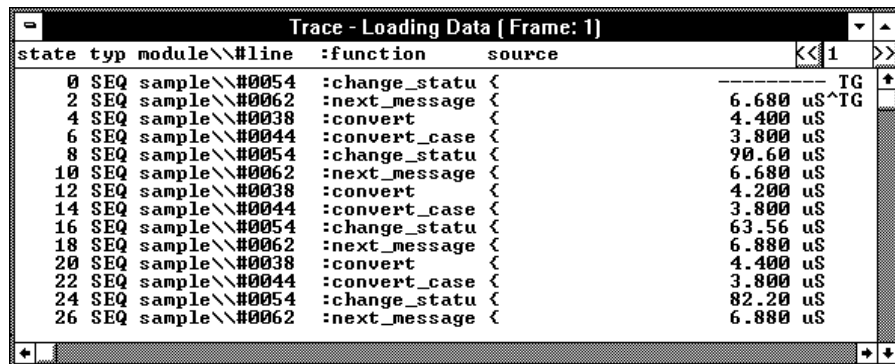
The command traces C function entry points only. It does not trace
execution for assembly language routines.

Note

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

Example

Function Flow Trace

```
┌─────────────────────────────────────────────────────────────────────┐
│  �       Trace - Loading Data [ Frame: 1]            ▼  ▲     │
├─────────────────────────────────────────────────────────────────────┤
│state typ module\\#line   :function      source          K<|1    >>   │
│     0 SEQ sample\\#0054   :change_statu {              --------- TG  ↑│
│     2 SEQ sample\\#0062   :next_message {              6.680 uS^TG   │
│     4 SEQ sample\\#0038   :convert      {              4.400 uS      │
│     6 SEQ sample\\#0044   :convert_case {              3.800 uS      │
│     8 SEQ sample\\#0054   :change_statu {              90.60 uS      │
│    10 SEQ sample\\#0062   :next_message {              6.680 uS      │
│    12 SEQ sample\\#0038   :convert      {              4.200 uS      │
│    14 SEQ sample\\#0044   :convert_case {              3.800 uS      │
│    16 SEQ sample\\#0054   :change_statu {              63.56 uS      │
│    18 SEQ sample\\#0062   :next_message {              6.880 uS      │
│    20 SEQ sample\\#0038   :convert      {              4.400 uS      │
│    22 SEQ sample\\#0044   :convert_case {              3.800 uS      │
│    24 SEQ sample\\#0054   :change_statu {              82.20 uS      │
│    26 SEQ sample\\#0062   :next_message {              6.880 uS      │
├─────────────────────────────────────────────────────────────────────┤
│ ← ▐                                                          →  ↓    │
└─────────────────────────────────────────────────────────────────────┘
```

To trace callers of a specified function

1 Double-click the function name in one of the debugger windows.

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

3 Choose the OK button.

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

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

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

Note

The analyzer may fail in tracing the caller due to prefetching in 68360. To avoid this failure, specify the function by a value of its address + 2.

Example

To trace the caller of "next_message":

Double-click "next_message".

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

```
┌─────────────────────────────────────────────────────────────────────┐
│ ▭                    M68360 RealTime C Debugger                      │
├─────────────────────────────────────────────────────────────────────┤
│  File   Execution   Breakpoint   Variable   Trace   RealTime   Assemble...   Settings │
│  Window   Help                             ┌─────────────────────────┐ │
│                                            │ Function Flow           │ │
│  ┌──────────────────────────Sourc────────│ Function Caller...       │▼│▲│
│  │Filename : c:\hp\rtc\m360\demo\sa│       │ Function Statement...    │ │
│  │                                 │       │ Variable Access...       │ │
│  │    #0046        for(mes = dat->│        │ Variable Break...        │▶)│↑│
│  │    #0047           if(*mes )=  │        │ Edit...                  │ │
│  │    #0048             *mes =    │        │                          │ │
│  │    #0049           else if(*me│        │ Trigger Store...         │ │
│  │    #0050             *mes =    │        │ Find Then Trigger...     │ │
│  │    #0051        }              │        │ Sequence...              │he│
│  │    #0052                       │        │                          │ │
│  │    #0053        change_status(int s│    │ Until Halt               │ │
│  │    #0054        {              │        │ Halt            Ctl+F7   │ │
│  │    #0055           if(st == ORIGIN│     │ Again           F7       │ │
│  │    #0056              return(CONU│     └─────────────────────────┘ │
│  │    #0057           else        │                                    │
│  │    #0058              return(ORIG│                                  │
│  │    #0059        }              │                                    │
│  │    #0060                       │                                    │
│  │    #0061        next_message(int id)    /* Change message to be     │
│  │                                                                    │
│  │←▮                                                             →│↓│  │
└─────────────────────────────────────────────────────────────────────┘
```

The Trace window becomes active and displays the trace results.

```
┌─────────────────────────────────────────────────────────────────────┐
│ ▭                   Trace - Loading Data [ Frame: 1]            ▼│▲│
├─────────────────────────────────────────────────────────────────────┤
│state typ module\\#line   :function     source          K<│        >>│
│                                                                      │
│┃   1 PRE sample\\#0058    :change_statu      return(ORI  prestore^TG │↑│
││    3     sample\\#0062    :next_message {                16.56 uS   │
││    4 PRE sample\\#0023    :main              message_id  prestore    │
││    6     sample\\#0062    :next_message {                5.360 uS   │
││    7 PRE sample\\#0059    :change_statu }                prestore    │
││    9     sample\\#0062    :next_message {                73.08 uS   │
││   10 PRE sample\\#0023    :main              message_id  prestore    │
││   12     sample\\#0062    :next_message {                5.360 uS   │
││   13 PRE sample\\#0059    :change_statu }                prestore    │
││   15     sample\\#0062    :next_message {                91.92 uS   │
││   16 PRE sample\\#0023    :main              message_id  prestore    │
││   18     sample\\#0062    :next_message {                5.360 uS   │
││   19 PRE sample\\#0058    :change_statu      return(ORI  prestore    │
││   21     sample\\#0062    :next_message {                80.80 uS   │
│                                                                      │
│←▮                                                             →│↓│  │
└─────────────────────────────────────────────────────────────────────┘
```

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

To trace execution within a specified function

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

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

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

Example

To trace execution within "next_message":

Double-click "next_message".

Choose the Trace→Function Statement... (ALT, T, S) command.
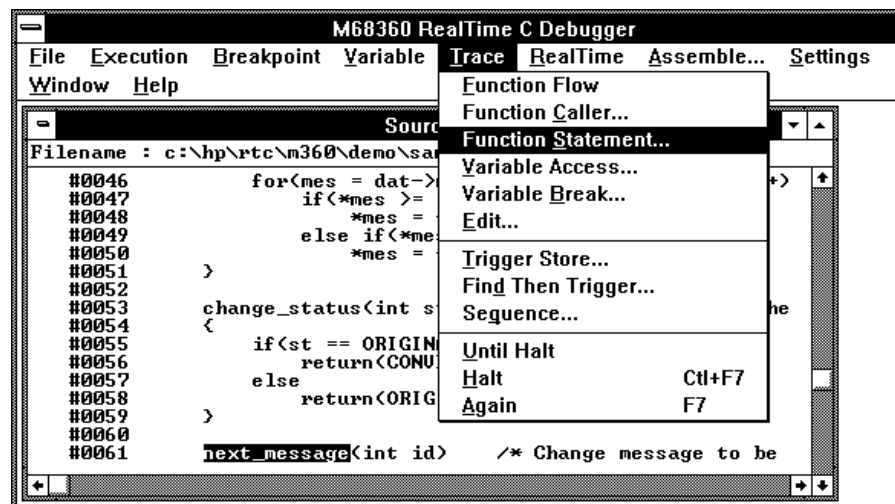
The Trace window becomes active and displays the results. You can see how prefetching affects tracing by choosing the Display→Mixed Mode (ALT, -, D, M) command from the Trace window's control menu.

To trace accesses to a specified variable

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

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

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

Example   To trace access to "message_id":

Double-click "message_id".

Choose the Trace→Variable Access... (ALT, T, V) command.
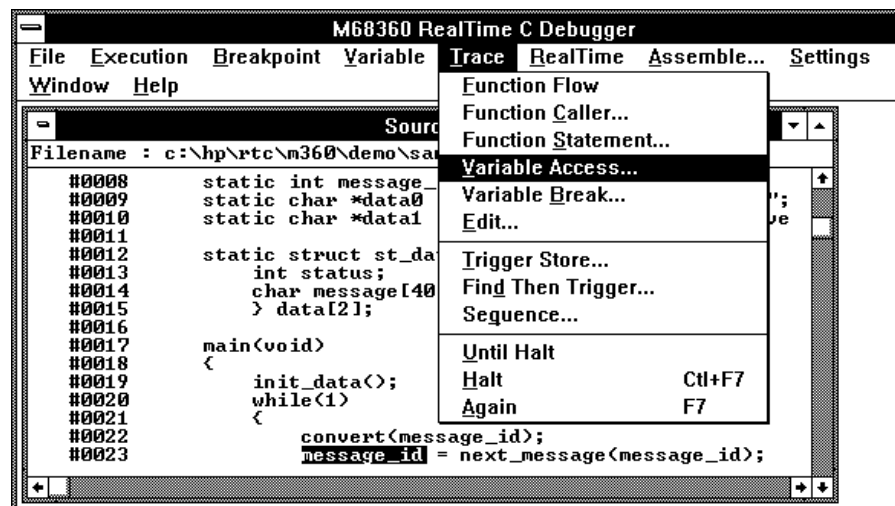
The Trace window becomes active and displays the trace results.

To trace before a particular variable value and break

1 Double-click the desired global variable.

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

3 Enter the value in the Value text box.

4 Choose the OK button.

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

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

Example

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

Double-click "message_id".

Choose the Trace→Variable Break... (ALT, T, B) command.
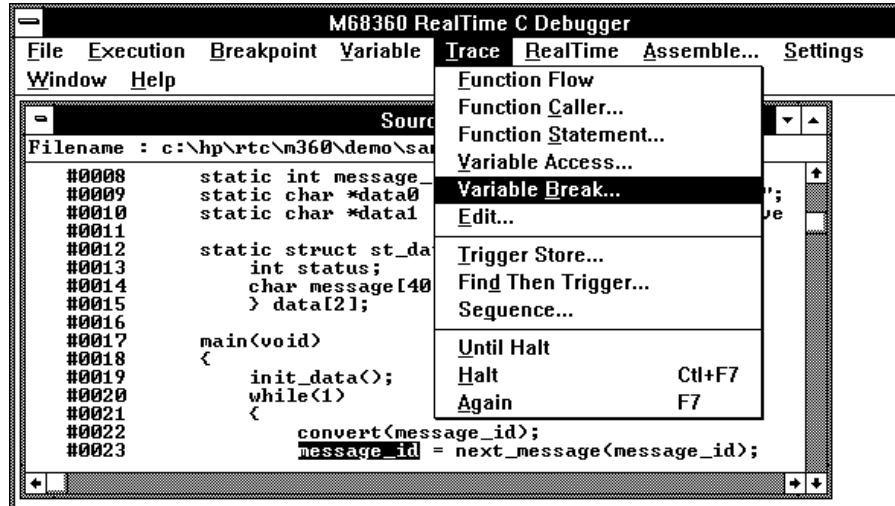
Enter "0" in the Value text box.
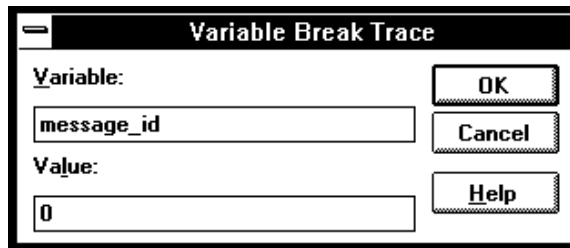
Choose the OK button.

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

To trace until the command is halted

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

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

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

To stop a running trace

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

The command is used to:

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

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

Stop a trace before the trace buffer becomes full.

To repeat the last trace

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

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

To display bus cycles

1 Place the cursor on the line from which you wish to display the bus cycles.

2 From the Trace window's control menu, choose the Display→Mixed Mode (ALT, -, D, M) command or the Display→Bus Cycle Only (ALT, -, D, C) command.

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

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

The display starts from the cursor-selected line.

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

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

```
┌─────────────────────────────────────────────────────────────────────┐
│ ▬           Trace - Loading Data [ Frame: 1]              ▼ ▲        │
│ state typ module\\#line  :function      source          <<│1   >>   │
│              sample\\#0054  :change_statu {        --------- TG  ▲  │
│     0 SEQ000005b8  4e560000  LINK.W   A6,#$0000    --------- TG     │
│     1 SEQ000005bc  4aae0008     supr prog rd long  0.200 uS         │
│              sample\\#0062  :next_message {         6.880 uS^TG     │
│     2 SEQ000005cc  4e560000  LINK.W   A6,#$0000    6.680 uS         │
│     3 SEQ000005d0  4aae0008     supr prog rd long  0.200 uS         │
│              sample\\#0038  :convert      {         4.400 uS        │
│     4 SEQ0000051c  4e560000  LINK.W   A6,#$0000    4.200 uS         │
│     5 SEQ00000520  48e73000     supr prog rd long  0.200 uS         │
│              sample\\#0044  :convert_case {         3.800 uS        │
│     6 SEQ00000580  4e560000  LINK.W   A6,#$0000    3.600 uS         │
│     7 SEQ00000584  206e0008     supr prog rd long  0.200 uS         │
│              sample\\#0054  :change_statu {         82.20 uS        │
│     8 SEQ000005b8  4e560000  LINK.W   A6,#$0000    82.00 uS         │
│ ◄ ▐                                                          ► ▼    │
└─────────────────────────────────────────────────────────────────────┘
```

To display absolute or relative counts

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

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

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

To change the disassembly of bus cycle data

1 From the Trace window's control menu, choose the Display→From
State... (ALT, -, D, F) command.

2 In the State text box, enter the number of the state where you want
disassembly to start.

3 Select either the High Word or Low Word option to specify which
16-bit word of the 32-bits of captured data to start disassembly from.

4 If the bus cycle data is being dequeued, enter the number of the
operand cycle state caused by the instruction cycle state you are
disassembling from.

5 Choose the OK button.

If you see disassembled bus cycle information in the Trace window that does
not look correct, you can use the Display→From State... (ALT, -, D, F)
command to change which state the disassembly starts from.
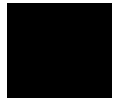
To display dequeued trace data

- From the Trace window's control menu, choose the
  Display→Options→Dequeue ON (ALT, -, D, O, O) command.

During 68360 program execution, operand cycles may not appear on the bus
immediately after the instruction cycles that cause them because the 68360
microprocessor prefetches instructions into (and executes them out of) an
instruction pileline. Consequently, when you trace microprocessor
execution, the captured bus cycle data may be difficult to read and
understand.

The Display→Options→Dequeue ON (ALT, -, D, O, O) command shuffles bus
cycle states in the Trace window so that operand cycles immediately follow
the instruction cycles that caused them. And, unexecuted instructions are
removed from the display.

To turn OFF dequeueing, choose the Display→Options→Dequeue OFF
(ALT, -, D, O, F) command from the Trace window's control menu.

If you are having problems tracing

- Check the emulator configuration. Choose the Settings→Emulator Config→Hardware... (ALT, S, E, H) command.

    - You may need to select Buffering of AS, DS, RW = Buffered.

    - You may need to select Clock 01 Mode = Buffered.

    Interaction problems between the emulator and the target system may affect tracing. These problems may be overcome by selecting proper buffering of the emulator signals.

- To obtain a trace, the analyzer must receive CLK01, and the clock must meet normal clock specifications.

    - Perhaps your target system is degrading the clock so that it does not meet specifications.

    - Perhaps the target system is interfering with proper operation of CLK01.

    Without CLK01, no trace can be taken by the analyzer.

Setting Up Custom Trace Specifications (Emulator Only)

This section shows you how:

- To set up a "Trigger Store" trace specification

- To set up a "Find Then Trigger" trace specification

- To set up a "Sequence" trace specification

- To edit a trace specification

- To trace "windows" of program execution

- To store the current trace specification

- To load a stored trace specification

Note

The analyzer traces unexecuted instructions due to prefetching in 68360.

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.

Automatic Long Alignment of Program Symbols in Trace Patterns

Because instructions are fetched from program memory space long words at a time (when there is a 32-bit data bus), it's difficult to symbolically set up a state qualifier pattern for program accesses of instructions on odd word boundaries.

To solve this problem, the debugger will automatically long align symbols in trace patterns when the "prog" status qualifier is used. Long alignment is accomplished by setting the two least significant address bits to zeros. Automatic long alignment will not occur if the symbol is part of an expression.

For example, if "main" is at address 2006H, the effective address 2004h would be used. In the examples that follow, only the first address is adjusted. The second example is not adjusted because it is in data space. The remaining examples are not adjusted because they use expressions rather than a symbol for the address value.

```
PATTERN INPUT                     EFFECTIVE PATTERN
a = A:main   D:100 S:prog         a = A:2004h D:100 S:prog
a = A:main   D:100 S:data         a = A:2006h D:100 S:data
a = A:2006h  D:100 S:prog         a = A:2006h D:100 S:prog
a = A:main+0 D:100 S:prog         a = A:2006h D:100 S:prog
a = A:main+1 D:100 S:prog         a = A:2007h D:100 S:prog
```

If you don't want long alignment to occur for an address within a trace pattern, simply make the symbol part of an expression (for example, main+0) or use the absolute address.

Program Symbols in Trace Results

Because of the way instructions are fetched when there is a 32-bit data bus, trace results can show symbolic information for the instruction on an even word boundary when the instruction on the odd word boundary was actually executed.

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.
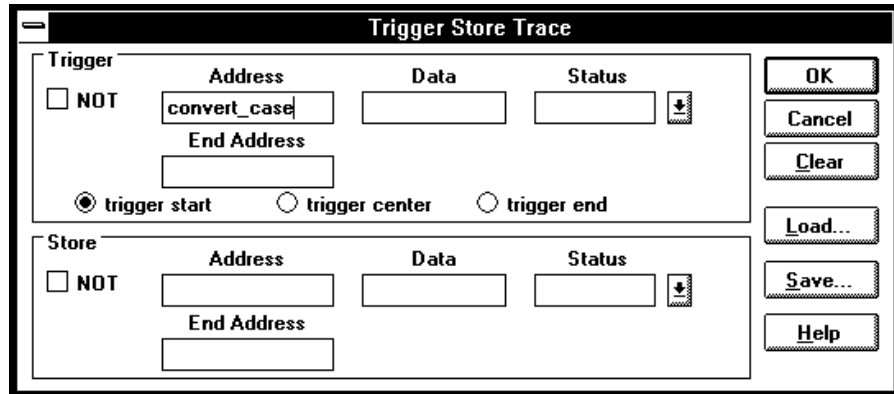
Choose the OK button.

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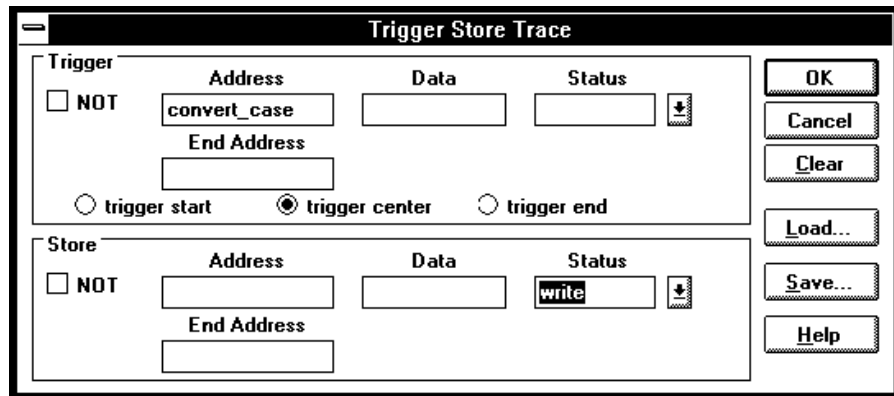
Example

To specify the trigger condition as any address in the range 1000h through 1fffh:
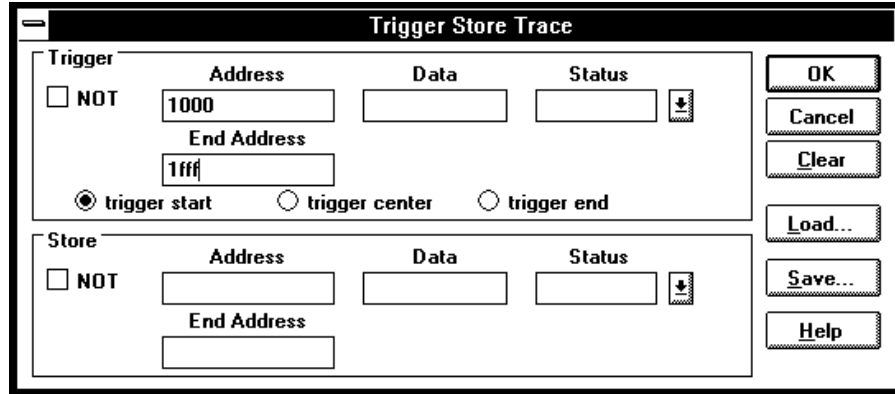
To set up a "Find Then Trigger" trace specification

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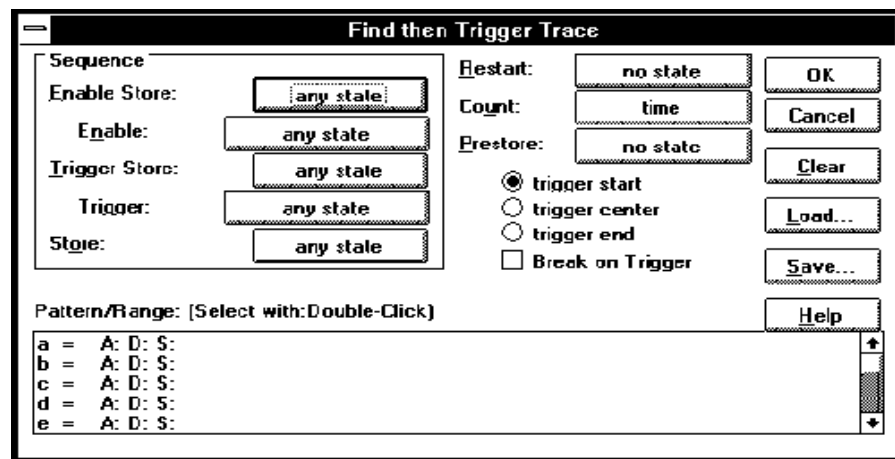
Choosing the enable, trigger, store, count, or prestore buttons opens a Condition dialog box that lets you select "any state", "no state", trace patterns

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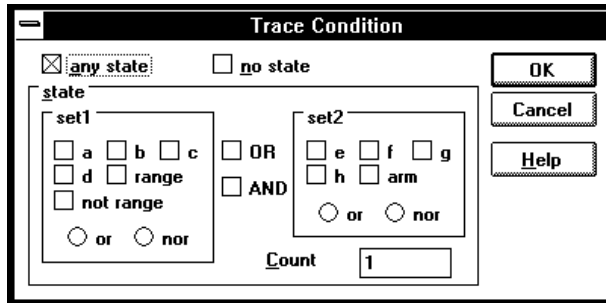
The range and pattern resources are defined by double-clicking on the resource name in the Pattern/Range list box.

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 (bottom of the Pattern/Range list box), the Trace Range dialog box is opened to let you select either the Address range or the Data range option and enter the minimum and maximum values in the range.
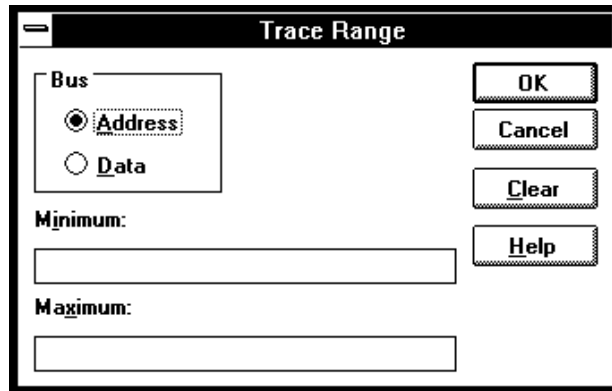
Example

To trace execution after the "convert_case" function:

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

Choose the Trigger button (default: any state).

Select "a".
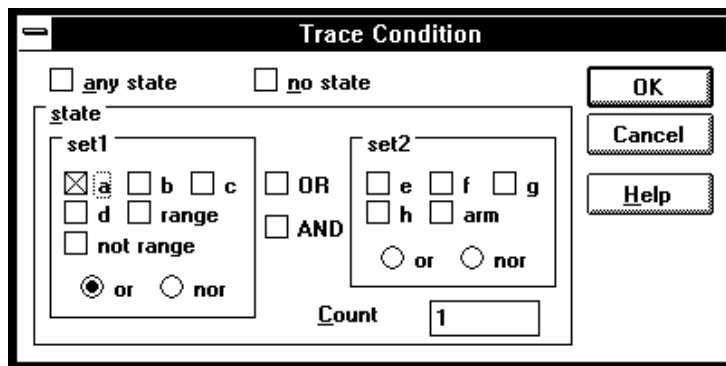
Choose the OK button.

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

Enter "convert_case" in the Address text box in the Trace Pattern dialog box.
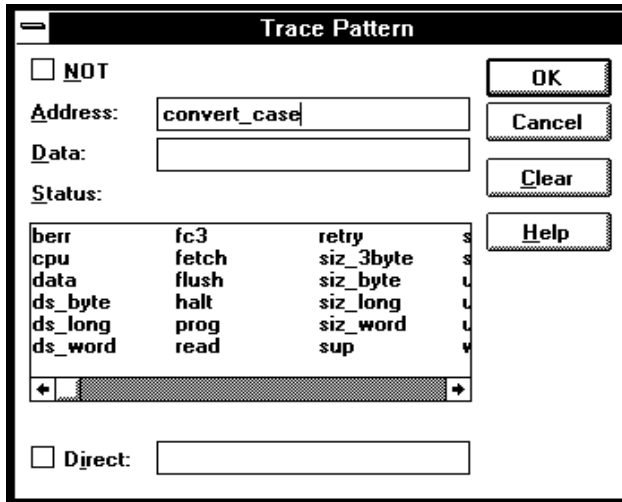
Choose the OK button in the Trace Pattern dialog box.
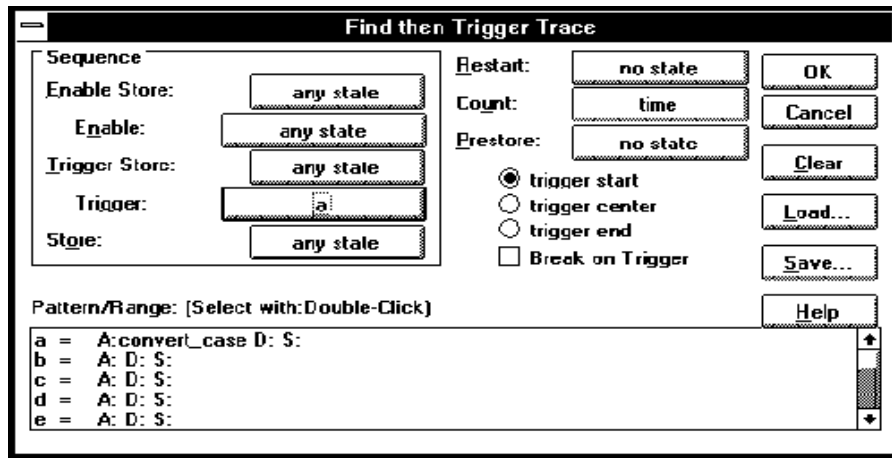
Choose the OK button in the Find then Trigger Trace dialog box.

Example

To trace about the "next_message" function when it follows the
"change_status" function and store all states after the "change_status"
function:
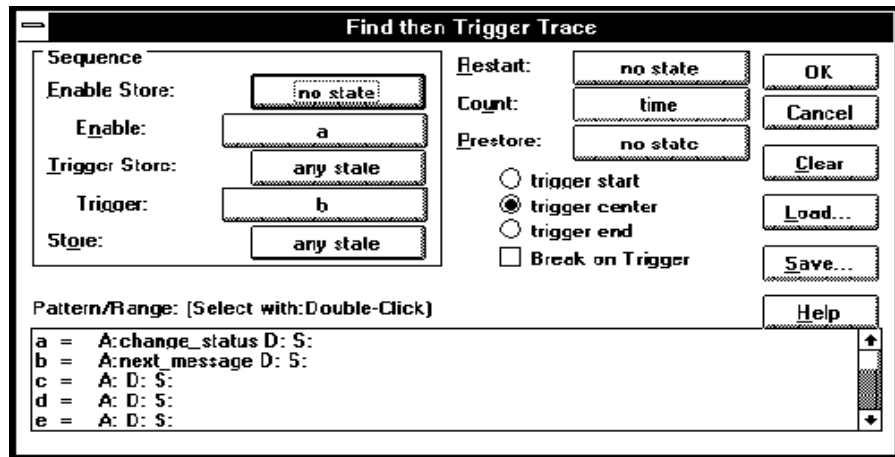
To set up a "Sequence" trace specification

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

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

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

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

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

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

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

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

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

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

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

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

The Trace→Sequence... (ALT, T, Q) command calls the Sequence Trace Setting dialog box, where you make the following trace specifications:
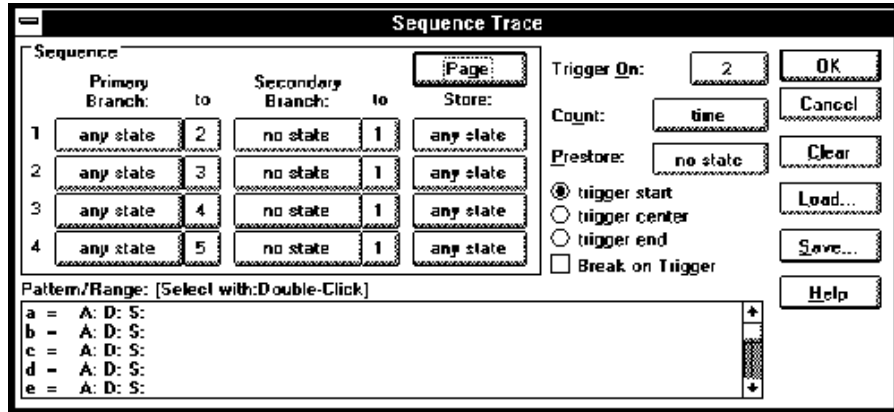
Choosing the primary branch, secondary branch, store, count, or prestore buttons opens a Condition dialog box that lets you select "any state", "no state", trace patterns "a" through "h", "range", or "arm" as the condition. Patterns "a" through "h", "range", and "arm" are grouped into two sets, and resources within a set may be combined using the "or" or "nor" logical operators. Resources in the two sets may be combined using the OR or AND logical operators.
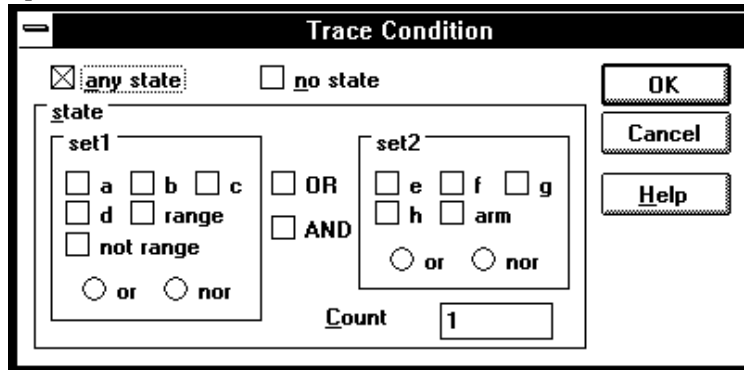
The range and pattern resources are defined by double-clicking on the resource name in the Pattern/Range list box.

If you double-click on a pattern name, the Trace Pattern dialog box is opened to let you specify address, data, and status values. By selecting the NOT check box, you can specify all states other than those identified by the address, data, and *status values*. The Direct check box lets you specify status values other than those that have been predefined.
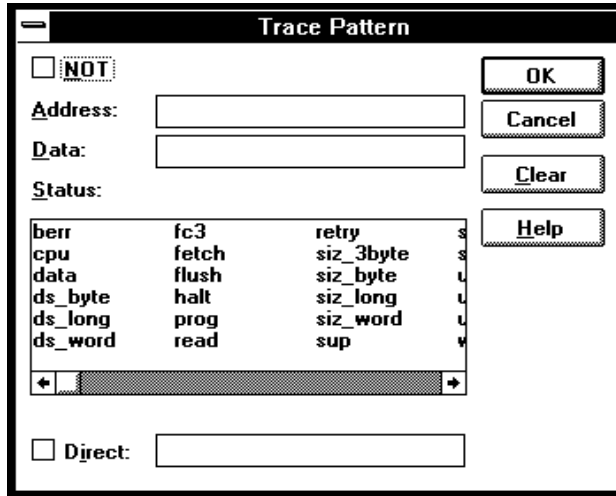
If you double-click on the range resource at the bottom of the Pattern/Range list box, the Trace Range dialog box is opened to let you select either the Address range option or the Data range option and enter the minimum and maximum values in the range.
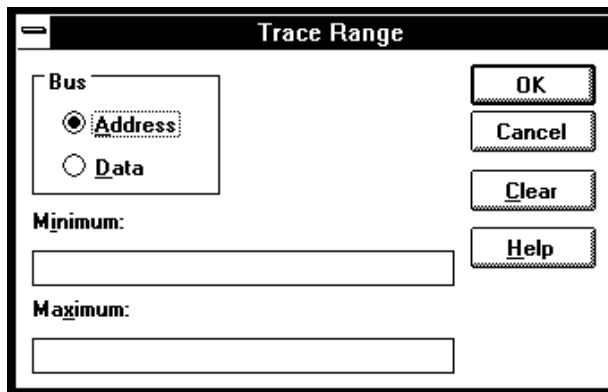
Example

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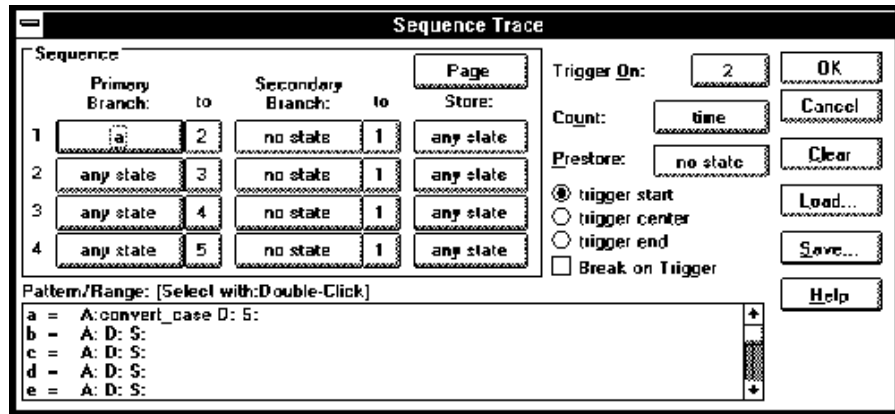
Example

To specify execution of "convert_case" and "next_message" as the trigger sequence:
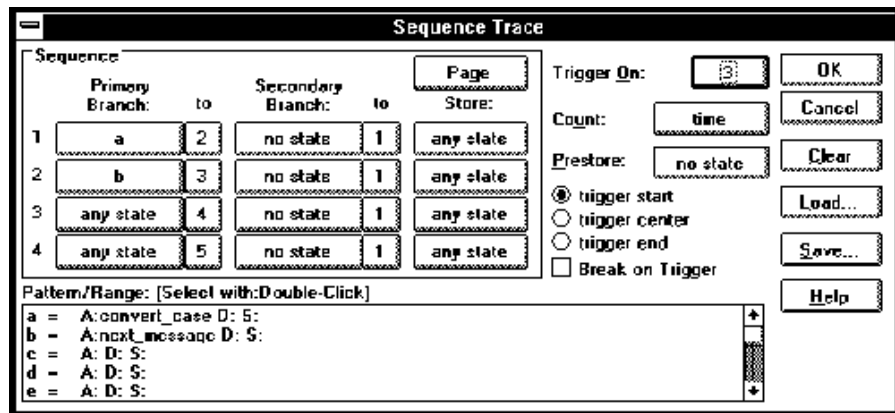
To edit a trace specification

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

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

3 Choose the OK button.

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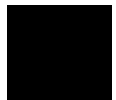
To trace "windows" of program execution

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

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

3 Choose the OK button.

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

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

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

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

WHILE STORING	FIND		ELSE ON	GO TO
no state	window enable state	1	no state	
any state	trigger state	2	window disable state	1
any state	window disable state	3	no state	
no state	window enable state	4		3
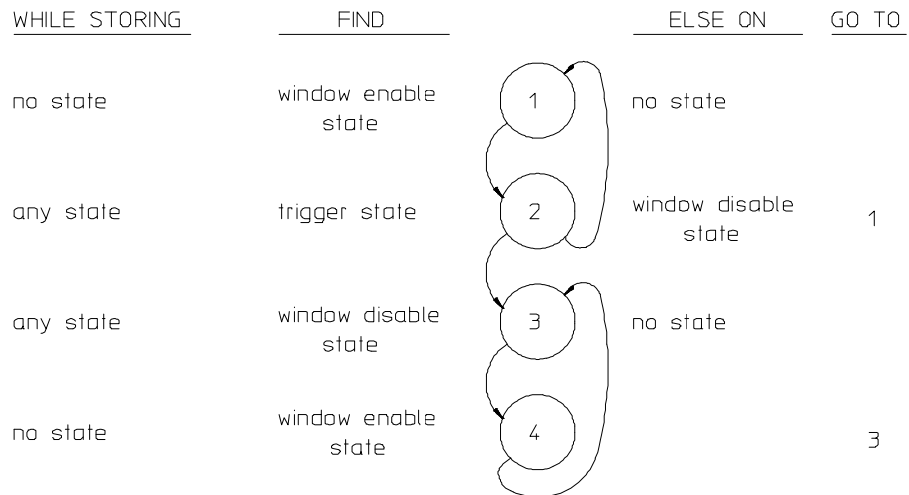
Notice that the order of the second sequence level pair is swapped. In sequence level 2, if the analyzer finds the trigger condition while searching for the window disable state, it will branch to sequence level 3 where it continues its search for the window disable state. After this, the analyzer will remain in sequence levels 3 and 4 until the trace memory is filled, completing the trace.

Example

To trace the window of code execution between lines 46 and 51 of the sample program, triggering on any state in the window:
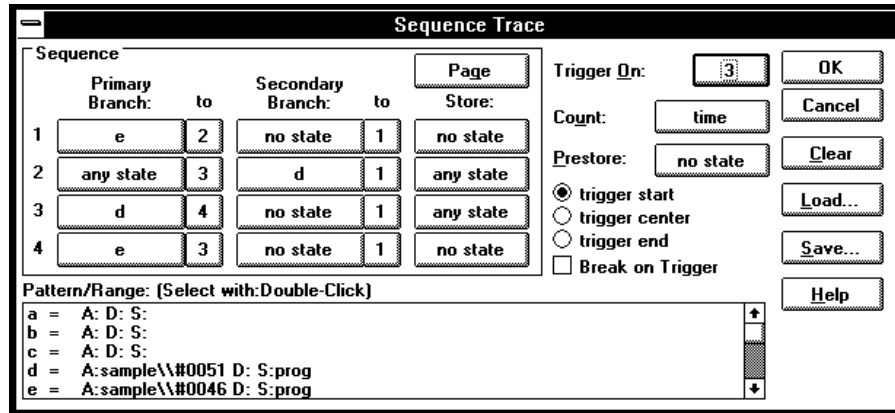
Notice that the analyzer triggers on the entry to sequence level 3. The primary branch condition in level 2 actually specifies the trigger condition.

To store the current trace specification

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

2 Choose the Save... button.

3 Specify the name of the trace specification file.

4 Choose the OK button.

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

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

To load a stored trace specification

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

2 Choose the Load... button.

3 Select the desired trace specification file.

4 Choose the OK button.

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

Programming Target Flash Memory (HP E3490A Software Probe Only)

This section shows you how:

- To program or erase a part

You can use the HP E3490A Software Probe to program flash memory on your target system.

Target system configuration

Prior to erasing or programming parts through the HP E3490A Software Probe, you must to establish any necessary options on the target system. This can include enabling the programming voltage (Vpp), disabling write protection, or properly setting the chip selects in the SIM. The chip selects in the SIM can be set through the configuration process. The documentation on the target system can provide information on such things as establishing the Vpp and disabling any target based write protection. The HP E3490A Software Probe accomplishes the FLASH ROM erasure and programming through standard target bus writes and reads.

Supported parts

The HP E3490A Software Probe can program standard Intel and AMD parts, and equivalent parts from other manufacturers. Supported algorithms include the Intel quick-pulse (or the identical AMD Flashrite), Intel Auto, AMD 5V embedded, and AMD 12V embedded algorithms.

Supported file sizes

There is no file size limit.
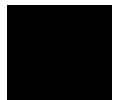
To program or erase a part

1 Create a file which contains the data to be flashed.

2 Set the necessary options (if any) on your target system to allow flash programming.

3 Choose the File→Flash Programming... command.

4 Select the appropriate options to program your flash memory.

5 Program the part by clicking on OK or Apply in the dialog box.

Part 3

Reference

Descriptions of the product in a dictionary or encyclopedia format.

Part 3

7
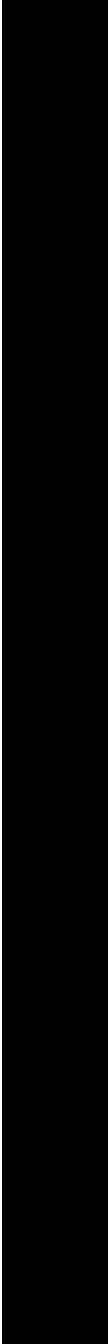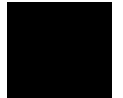
Command File and Macro Command
Summary

Command File and Macro Command Summary

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

The characters in parentheses can be ignored for shortcut entry.

Run Control Commands

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

Variable and Memory Commands

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

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

Breakpoint Commands

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

Window Open/Close Command

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

Configuration Command

Command	Param_1	Param_2	Param_3	Param_4	Operation
MON(ITOR)	STA(RT)				Starting monitor
MON(ITOR)	mon-item	mon-ans			Setting up monitor
MON(ITOR)	END				Ending monitor
CON(FIG)	STA(RT)				Starting configuration
CON(FIG)	config-item	config-ans			Executing configuration
CFGBDM	config-item	config-ans			Executing BDM configuration
CON(FIG)	END				Ending configuration
MAP	STA(RT)				Starting mapping
MAP	addr-range	memtype	func-code	attribute	Executing mapping
MAP	OTHER	memtype	func-code		Mapping OTHER area
MAP	END				Ending mapping
MOD(E)	MNE(MONIC)	ON			Enabling Mnemonic display
MOD(E)	MNE(MONIC)	OFF			Enabling Source display
MOD(E)	REA(LTIME)	ON			Enabling real-time mode
MOD(E)	REA(LTIME)	OFF			Disabling real-time mode
MOD(E)	IOG(UARD)	ON			Enabling I/O guard
MOD(E)	IOG(UARD)	OFF			Disabling I/O guard
MOD(E)	MEM(ORYPOLL)	ON			Enabling Memory polling
MOD(E)	MEM(ORYPOLL)	OFF			Disabling Memory polling
MOD(E)	WAT(CHPOLL)	ON			Enabling WatchPoint polling
MOD(E)	WAT(CHPOLL)	OFF			Disabling WatchPoint polling
MOD(E)	LOG	ON			Enabling log file output
MOD(E)	LOG	OFF			Disabling log file output
MOD(E)	BNC	IN			Setting BNC input
MOD(E)	BNC	OUT			Setting BNC output
MOD(E)	SYM(BOLCASE)	ON			Case sensitive symbol search
MOD(E)	SYM(BOLCASE)	OFF			Case insensitive sym. search
MOD(E)	TRA(CE)	DIS(PLAY)	FRO(M)	state-num	Change bus cycle disassembly
MOD(E)	TRA(CE)	DIS(PLAY)	HIG(WORD)/LOW(WORD)		Word to disassemble from
MOD(E)	TRA(CE)	DIS(PLAY)	ALI(GN)	state-num	Operand state in dequeueing

```
MOD(E)       TRA(CE)       DIS(PLAY)      DEQ(UEUE)      Dequeue bus cycle data
MOD(E)       TRA(CE)       DIS(PLAY)      NOD(EQUEUE)    Display bus data as captured
MOD(E)       DOW(NLOAD)    ERR(ABORT)                    Error causes load abort
MOD(E)       DOW(NLOAD)    NOE(RRABORT)                  Load continues after error
MOD(E)       SOU(RCE)      ASK(PATH)                     Prompt for source paths
MOD(E)       SOU(RCE)      NOA(SKPATH)                   Don't prompt for source
                                                         paths
MOD(E)       TRACECLOCK    BACKGROUND                    Trace background cycles
MOD(E)       TRACECLOCK    BOTH                          Trace all processor cycles
MOD(E)       TRACECLOCK    USER                          Trace user program cycles
CFG(INFO)    INF(O)        0-9                           Display configuration info.
CFG(INFO)    SYN(C)        0-3                           Synchronize SIM & EMSIM reg.
```

File Command

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

Trace Commands

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

Symbol Window Commands

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

Command File Control Command

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

Miscellaneous Commands

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

Parameters

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

8
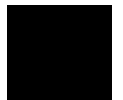
Expressions in Commands

Expressions in Commands

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

- Numeric constants (hexadecimal, decimal, octal, or binary values).

- Symbols (identifiers).

- Function codes.

- C operators (pointers, arrays, structures, unions, unary minus operators) and parentheses (specifying the order of operator evaluation).

Numeric Constants

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

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

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

                            Alphanumeric strings starting with any of '0' through '9', ending with 'H' or 'h', and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 12345678H, 0FFFF0000h).

                            Alphanumeric strings starting with any of '0' through '9' and consisting of any of '0' through '9', 'A' through 'F', or 'a' through 'f' (for example: 12345678, 0FFFF0000).

                            Hexadecimal strings starting with alphabetical characters must be preceded by 0. For example, FF40H must be entered as 0FF40H.

Decimal   Numeric strings consisting of any of '0' through '9' and ending with 'T' or 't' (for example: 128T, 1000t).

Octal   Numeric strings consisting of any of '0' through '7' and ending with 'O' or 'o' (not zero) (for example: 200o, 377O).

Binary   Numeric strings consisting of '0' or '1' and ending with 'Y' or 'y' (for example: 10000000y, 11001011Y).

Don't Care   Numeric strings containing 'X' or 'x' values. All numeric strings must begin with a numeric value. For example, x1x0y must be entered as 0x1x0y.

Symbols

The debugger expressions support the following symbols (identifiers):

- Symbols defined in C source code.

- Symbols defined in assembly language source code.

- Symbols added with the Symbol window control menu's User defined→Add... (ALT, -, U, A) command.

- Line number symbols.

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

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

Module Name

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

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

Symbol Name

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

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

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

Symbol address   &symbol_name

Symbol data   *symbol_name

Format Specification

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

String   s

Decimal   d (current size), d8 (8 bit), d16 (16 bit), d32 (32 bit)

Unsigned   u (current size), u8 (8 bit), u16 (16 bit), u32 (32 bit)
decimal

Hexadecimal   x (current size), x8 (8 bit), x16 (16 bit), x32 (32 bit)

Examples

Some example symbol expressions are shown below:

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

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

```
sample\\#22
```

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

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

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

dat→message,s

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

dat→message,x32

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

sample\\data[1].status,d32

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

&data[0]

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

*1000

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

Function Codes

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

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

- Specifying addresses in trace commands (because address qualifiers are compared with values captured on the address bus -- function code information is captured as part of the bus cycle status).

- Referring to a program counter address (because the function code is determined by the Supervisor/User status flag bit).

C Operators

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

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

9

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

- Trace→Function Statement... (ALT, T, S)
- Trace→Variable Access... (ALT, T, V)
- Trace→Variable Break... (ALT, T, B)
- Trace→Edit... (ALT, T, E)
- Trace→Trigger Store... (ALT, T, T)
- Trace→Find Then Trigger... (ALT, T, D)
- Trace→Sequence... (ALT, T, Q)
- Trace→Until Halt (ALT, T, U)
- Trace→Halt (ALT, T, H)
- Trace→Again (ALT, T, A)
- RealTime→Monitor Intrusion→Disallowed (ALT, R, T, D)
- RealTime→Monitor Intrusion→Allowed (ALT, R, T, A)
- RealTime→I/O Polling→ON (ALT, R, I, O)
- RealTime→I/O Polling→OFF (ALT, R, I, F)
- RealTime→Watchpoint Polling→ON (ALT, R, W, O)
- RealTime→Watchpoint Polling→OFF (ALT, R, W, F)
- RealTime→Memory Polling→ON (ALT, R, M, O)
- RealTime→Memory Polling→OFF (ALT, R, M, F)
- Assemble... (ALT, A)
- Settings→Emulator Config→Hardware... (ALT, S, E, H)
- Settings→Emulator Config→Memory Map... (ALT, S, E, M)
- Settings→Emulator Config→Monitor... (ALT, S, E, O)
- Settings→Emulator Config→Information... (ALT, S, E, I)
- 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→Trace Cycles→User (ALT, S, X, T, U)
- Settings→Extended→Trace Cycles→Monitor (ALT, S, X, T, M)
- Settings→Extended→Trace Cycles→Both (ALT, S, X, T, B)
- Settings→Extended→Load Error Abort→ON (ALT, S, X, L, O)
- Settings→Extended→Load Error Abort→OFF (ALT, S, X, L, F)
- Settings→Extended→Source Path Query→ON (ALT, S, X, S, O)
- Settings→Extended→Source Path Query→OFF (ALT, S, X, S, F)
- Window→Cascade (ALT, W, C)

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

File→Load Object... (ALT, F, L)

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

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

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

Microtec MCC68K Compiler

Microtec ASM68K Assembler

Microtec LNK68K Linker

HP AxLS CC68000 Compiler

HP AxLS AS68K Assembler

HP AxLS LD68K Linker

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

Load Object File Dialog Box

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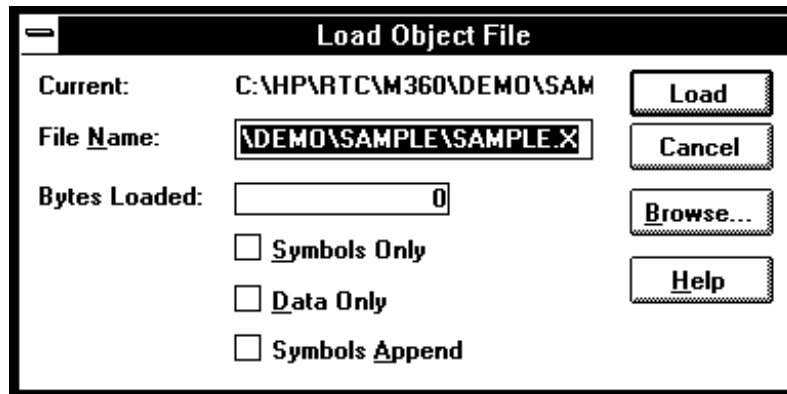
Current	Shows the currently loaded object file.
File Name	Specifies the object file to be loaded. The system defaults the file extension to ".x".
Fcode	Assigns any of the *function codes* to the destination memory area.
Bytes Loaded	Displays the loaded data in Kbytes.
Symbols Only	Loads only the symbolic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

Command File Command

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

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

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

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

See Also

"To load user programs" in the "Loading and Displaying Programs" section of
the "Debugging Programs" chapter.
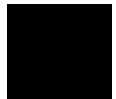
File→Flash Programming... (ALT, F, F)

(HP E3490A Software Probe Only.)

Programs flash memory parts in the target system.

This dialog box provides a mechanism to control programming and erasing of flash memory on the target system. It does this by issuing background cycles on the target system bus. The target system's SIM registers and other hardware must be set up to respond to bus cycles for the flash memory prior to executing any FLASH operations.
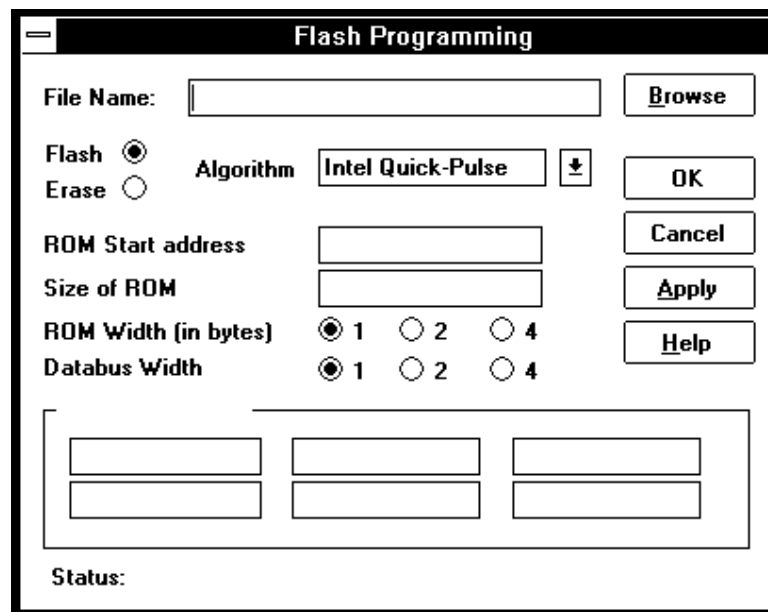
Flash	Choose Flash to download code into the flash device from an absolute, Motorola S-Record, or Intel Hex file.
Erase	Choose Erase to erase all or selected sectors of the flash device, depending on the capabilities of the device.
Algorithm	Lets you select select the algorithm to use for the flash device. The algorithm is selected from one of four currently supported. If you do not know which algorithm to use for your part, consult the data sheet for your part, or see if it is

in the partial list of parts and their algorithms at the end of this help. The four currently supported algorithms are:

AMD 12V Embedded
AMD 5V Embedded
Intel Auto
Intel Quick-Pulse (or AMD Flashrite)

ROM Start address
Use this field to enter the starting address on the target bus for the Flash ROM device. You may enter a number or symbol from your target program.

Size of ROM
Use this field to enter the size (in bytes) of the Flash ROM device. You may enter a number or symbol from your target program.

ROM Width (in bytes)
Use these buttons to select the width of the Flash ROM device itself.

File Name
Use this field to enter the name of the file which contains the data to be programmed into the Flash ROM device. This field is used only for the Program operation.

Browse
Opens a file selection dialog to pick a file.

Sector Addresses
This section will be active if you have selected the Erase operation. The addresses for the sectors may be specified as numbers or using program symbols. You may specify up to six sectors at once. To erase more than six, simply enter the additional sectors after applying the erase function with the first six. For the Intel QuickPulse, AMD 12V Embedded, and AMD 5V Sector algorithms, the entire part will be erased if you do not specify any sectors.

Status
Displays status information and error messages.

OK
Applies the values specified, and closes the dialog box.

Cancel
Closes the dialog box.

Apply
Applies the values specified, without closing the dialog box.
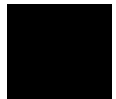
FLASH Device List

Following is a partial list of FLASH devices family, part designation, package, and algorithm to use. Be sure to consult your manufacturer's data sheet to verify the algorithm selection before programming.

```
FAMILY  Part  Package  Algorithm
 ------------------------ ------------  ------------  -----------------
 AMD 12V Bulk Erase  Am28F256  32-pin PLCC  Intel Quick-Pulse
   Am28F512  32-pin PLCC  Intel Quick-Pulse
   Am28F010  32-pin PLCC  Intel Quick-Pulse
   Am28F020  32-pin PLCC  Intel Quick-Pulse
   Am28F256A  32-pin PLCC  AMD 12V Embedded
   Am28F512A  32-pin PLCC  AMD 12V Embedded
   Am28F010A  32-pin PLCC  AMD 12V Embedded
   Am28F020A  32-pin PLCC  AMD 12V Embedded
 ------------------------ ------------  ------------  -----------------
 AMD 5V only  Secor erase  Am29F010  32-pin PLCC  AMD 5V Embedded
   Am29F100  AMD 5V Embedded
   Am29F200  AMD 5V Embedded
   Am29F040  44-pin SOP  AMD 5V Embedded
   Am29F400  AMD 5V Embedded
   Am29F016  AMD 5V Embedded
 ------------------------ ------------  ------------  -----------------
 Intel FlashFile  28F032SA  Intel Auto
   28F016SA  Intel Auto
   28F008SA  Intel Auto
 ------------------------ ------------  ------------  -----------------
 Intel Boot Block  28F400BX  44-pin SO  Intel Auto
   28F200B  Intel Auto
   28F001B  Intel Auto
 ------------------------ ------------  ------------  -----------------
 Intel Bulk erase  28F020  32-pin PLCC  Intel Quick-Pulse
   28F010  32-pin PLCC  Intel Quick-Pulse
   28F512  32-pin PLCC  Intel Quick-Pulse
   28F256A  32-pin PLCC  Intel Quick-Pulse
 ------------------------ ------------  ------------  -----------------
 Mitsubishi  M5M28F101  32-pin PLC  Intel Quick-Pulse
   M5M28F10  Intel Quick-Pulse
   M5M28F400  44-pin SOP  Intel Quick-Pulse
   M5M28F016  Intel Auto
 ------------------------ ------------  ------------  -----------------
 TI  TMS28F010A  32-pin PLC  Intel Quick-Pulse
   TMS28F010  32-pin PLC  Intel Quick-Pulse
   TMS28F512A  32-pin PLC  Intel Quick-Pulse
   TMS28F210  Intel Quick-Pulse
   TMS28F400  Intel Auto
 ------------------------ ------------  ------------  -----------------
 Hitachi  28F4001  32-pin DIP  Intel Quick-Pulse
   28F101  Intel Quick-Pulse
 ------------------------ ------------  ------------  -----------------
 SGS-Thomson  28F410  44-pin SOP  Intel Auto
   28F420  44-pin SOP  Intel Auto
 ------------------------ ------------  ------------  -----------------

 NOT SUPPORTED
 Atmel
 Hitachi  HN28F101
 Toshiba
 ------------------------ ------------  ------------  -----------------
```

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.
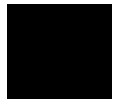
File→Run Cmd File... (ALT, F, R)

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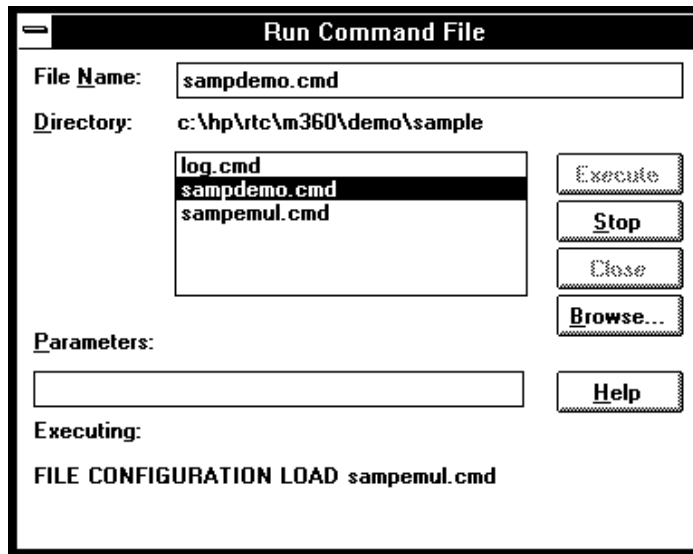
File Name   Lets you enter the name of the command file to be executed.

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 (emulator only).

- Monitor configuration settings (emulator only).

- EMSIM register values (HP E3490A Software Probe only).

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 (emulator only).

- Monitor configuration settings (emulator only).

- EMSIM register values (HP E3490A Software Probe only).

Command File Command

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

See Also

"To save the current emulator configuration" in the "Saving and Loading
Configurations" section of the "Configuring the Emulator" chapter.
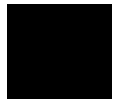
File→Copy Destination... (ALT, F, P)

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

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

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

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

Command File Command

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

See Also

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

File→Exit (ALT, F, X)

Exits the debugger.

Command File Command

`QUI(T)`

See Also

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

File→Exit HW Locked (ALT, F, H)

File→Exit HW Locked (ALT, F, H)

Exits the debugger and locks the emulator hardware.

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

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

If you have any breakpoints set when you exit the debugger, you will have to reset the breakpoints when you restart the debugger. All breakpoints are deleted when RTC is exited.

Command File Command

QUI(T) LOC(KED)

See Also

Settings→Communication... (ALT, S, C)

File Selection Dialog Boxes

File selection dialog boxes are used with several of the debugger commands. An example of a file selection dialog box is shown below.
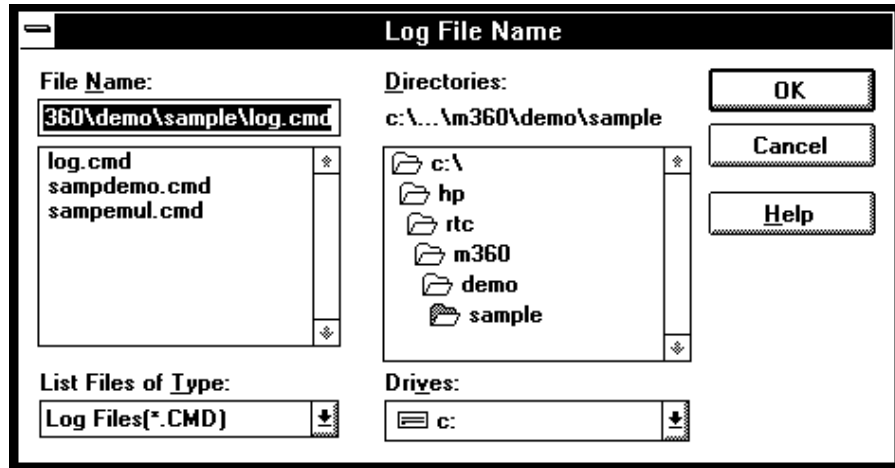
File Name   You can select the name of the file from the list box and edit it in the text box.

List Files of   Lets you choose the filter for files shown in the File Name
Type   list box.

Directories   You can select the directory from the list box. The selected directory is shown above the list box.

Drives   Lets you select the drive name whose directories are shown in the Directories list box.

OK   Selects the named file and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

Help   If this button is available, it opens a help window for viewing the associated help information.

Execution→Run (F5), (ALT, E, U)

Runs the program from the current program counter address.

Command File Command

RUN

Execution→Run to Cursor (ALT, E, C)

Runs from the current program counter address up to the Source window line that contains the cursor.

This command sets a breakpoint at the cursor-selected source line and runs from the current program counter address; therefore, it cannot be used when programs are in target system ROM.

If the cursor-selected source line is not reached within the number of milliseconds specified by StepTimerLen in the B3627.INI file, a dialog box appears from which you can cancel the command. When the Stop button is chosen, program execution stops, the breakpoint is deleted, and the processor continues RUNNING IN USER PROGRAM.

Command File Command

```
COM(E) address
```

See Also

"To run the program until the specified line" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Run to Caller (ALT, E, T)

Executes the user program until the current function returns to its caller.

Because this command determines the address at which to stop execution based on stack frame data and object file function information, the following restrictions are imposed:

- A function cannot properly return immediately after its entry point because the stack frame for the function has not yet been generated. Use the Step command to single-step the function before using the Execution→Run to Caller (ALT, E, T) command.

- An assembly language routine cannot properly return, even it follows C function call conventions, because there is no function information in the object file.

- An interrupt function cannot properly return because it uses a stack in a different fashion from standard functions.

Command File Command

RET(URN)

See Also

"To run the program until the current function return" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Run... (ALT, E, R)

Executes the user program starting from the specified address.

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

Note

If you try to run from an address whose symbol is START, STA, RESET, or RES (or any upper- or lower-case variation), the debugger instead runs from the start address or reset address, respectively, because these are the keywords used with the RUN command. To fix this problem, use START+0, STA+0, RESET+0, or RES+0 to force the symbol to be evaluated as an address.

Run Dialog Box

Choosing the Execution→Run... (ALT, E, R) command opens the following dialog box:
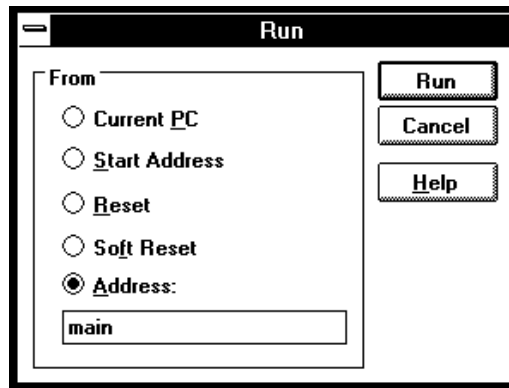
Current PC   Specifies that the program run from the current program counter address.

Start Address   Specifies that the program run from the *transfer address* defined in the object file.

257

User Reset	Resets the emulation processor and lets the emulator run and fetch its stack pointer and program counter value from memory.
Soft Reset	Pulses the 68360 soft reset line to cause a soft reset.
Address	Lets you enter the address from which to run. Because the function code is determined from the memory map, do not include one with the address.
Run	Initiates program execution from the specified address, then close the dialog box.
Cancel	Cancels the command and closes the dialog box.

Command File Command

```
RUN
```
Executes the user program from the current program counter address.

```
RUN STA(RT)
```
Executes the user program from the transfer address defined in the object file.

```
RUN RES(ET)
```
Drives the target reset line and begins executing from the contents of exception vector 0.

```
RUN SOF(TRESET)
```
Causes a soft reset.

```
RUN address
```
Executes the user program from the specified address.

See Also

"To run the program from a specified address" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Single Step (F2), (ALT, E, N)

Executes a single instruction or source line at the current program counter address.

A single source line is executed when in the source only display mode, unless no source is available or an assembly language program is loaded; in these cases, a single assembly language instruction is executed.

When in the mnemonic mixed display mode, a single assembly language instruction is executed.

Command File Command

STE(P)

See Also

"To step a single line or instruction" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Step Over (ALT, E, O)
Execution→Step... (ALT, E, S)

Execution→Step Over (F3), (ALT, E, O)

Executes a single instruction or source line at the current program counter except when the instruction or source line makes a subroutine or function call, in which case the entire subroutine or function is executed.

This command is the same as the Execution→Single Step (ALT, E, N) command except when the source line contains a function call or the assembly instruction makes a subroutine call (with the BSR or JSR instructions). In these cases, the entire function or subroutine is executed.

Command File Command

OVE(R)

See Also

"To step over a function" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Execution→Step... (ALT, E, S)

Single-steps the specified number of instructions or source lines, starting from the specified address.

Single source lines are executed when in the source only display mode, unless no source is available or an assembly language program is loaded; in these cases, single assembly language instructions are executed.

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

Note

If you try to step from an address whose symbol is START or STA (or any upper- or lower-case variation), the debugger instead steps from the start address because these are the keywords used with the STEP and OVER commands. To fix this problem, use START+0 or STA+0 to force the symbol to be evaluated as an address.

Step Dialog Box

Choosing the Execution→Step... (ALT, E, S) command opens the following dialog box:

Current PC   Specifies that stepping start from the current program counter address.

Start Address   Specifies that stepping start from the start address or *transfer address*.

Address   Lets you enter the address from which to single-step.

Count   Indicates the step count. The count decrements by one for every step and stops at 0.

Over   If the source line to be executed contains a function call or the assembly language instruction to be executed contains a subroutine call, this option specifies that the entire function or subroutine be executed.

Follow PC   If you check the Follow PC box, stepping will provide more detail because it will follow the PC for each step, and update the Source window after each step. Leaving this box unchecked speeds the stepping process; the steps will be counted, but the content of the Source window will not be updated until stepping is completed.

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

Command File Command

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

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

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

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

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

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

See Also

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

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

Execution→Break (F4), (ALT, E, B)

Stop user program execution and break into the monitor.

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

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

Command File Command

BRE(AK)

See Also

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

Execution→Reset (ALT, E, E)

Resets the emulation microprocessor.

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

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

Note that if the RealTime→Monitor Intrusion→Allowed (ALT, R, T, A) command is chosen, the emulation microprocessor may switch immediately from reset to running in monitor, for example, to update the contents of a register window.

Command File Command

RES(ET)

See Also

"To reset the processor" in the "Stepping, Running, and Stopping" section of the "Debugging Programs" chapter.

Breakpoint→Set at Cursor (ALT, B, S)

Sets a breakpoint at the cursor-selected address in the Source window.

The Breakpoint→Set at Cursor (ALT, B, S) command replaces the original instruction at the specified address with a BGND instruction, and the breakpoint marker, "BP", appears on that line.

When a breakpoint is hit (that is, when the BGND instruction is executed), program execution stops immediately before the instruction or source code line at which the breakpoint is set, and the emulator enters its monitor state.

A set breakpoint remains active until it is deleted. When you delete the breakpoint, the original instruction is restored in the user program.

Because breakpoints are set by replacing program opcodes with breakpoint instructions, they cannot be set in programs stored in target system ROM. In addition, breakpoints do not function properly when set at addresses where no opcode is found.

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

Command File Command

```
BP SET address
```

See Also

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

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

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

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

Command File Command

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

See Also

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

Breakpoint→Edit... (ALT, B, E)

Breakpoint→Set Macro... (ALT, B, M)

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

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

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

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

Because break macros use breakpoints, they cannot be set at addresses in target system ROM.

Additional commands can be added to existing break macros as follows:

- When a source code line or disassembled instruction is cursor-selected, the additional command is inserted at the top of the list of commands.

- When a macro command line is cursor-selected, the additional command is inserted immediately following the cursor-selected command.

Breakpoint Edit Dialog Box

Choosing the Breakpoint→Set Macro... (ALT, B, M) command opens the
following dialog box:
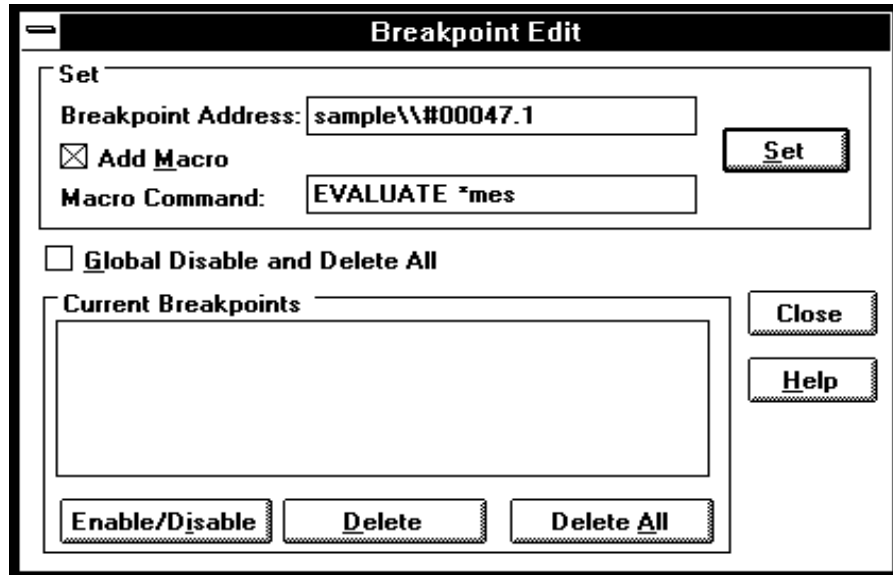
Breakpoint Address	Displays the specified line number or address followed by a decimal point and the break macro line number.
Add Macro	Activates the Macro Command text box.
Macro Command	Specifies the command to be added to the break macro.
Set	Inserts the specified macro command at the location immediately preceding the specified source line or address, or inserts the macro command at the location immediately following the specified break macro line.
	Two or more commands can be associated with a break macro by entering the first command and choosing Set, then entering the second command and choosing Set, and so on. Commands execute in the order of their entry.

Global Disable and Delete All	Disables and deletes all current breakpoints and break macros.
Current Breakpoints	Displays the addresses and line numbers of the current breakpoints and break macros. Allows you to select breakpoints or break macros to be deleted.
Enable/Disable	Enable/Disable the selected breakpoint and break macro.
Delete	Deletes the selected breakpoints or break macros from the Current Breakpoints list box.
Delete All	Deletes all breakpoints and break macros from the Current Breakpoints list box.
Close	Closes the dialog box.

Command File Command

```
BM SET address command
```

See Also

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

Breakpoint→Delete Macro (ALT, B, L)

Removes the break macro set at the cursor-indicated address in the Source window.

This command is only applicable to lines that contain "BP" markers (which indicate set breakpoints) or break macro lines.

When a source code line is cursor-selected, this command removes the breakpoint and all the macros commands set at the line.

When a break macro line is cursor-selected, this command removes the single macro command at the line.

Command File Command

```
BM DEL(ETE) address
```

See Also

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

Breakpoint→Edit... (ALT, B, E)

Breakpoint→Edit... (ALT, B, E)

Lets you set, list, or delete breakpoints and break macros. Breakpoints are always globally enabled on initial entry into the RTC interface.

Breakpoint Edit Dialog Box

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

```
┌──────────────────────────────────────────────────┐
│ ▬           Breakpoint Edit                       │
│ ┌─Set──────────────────────────────────────────┐ │
│ │ Breakpoint Address: sample\\#0032            │ │
│ │ ☒ Add Macro                        ┌───────┐ │ │
│ │ Macro Command:                     │  Set  │ │ │
│ │                                    └───────┘ │ │
│ └──────────────────────────────────────────────┘ │
│ ☐ Global Disable and Delete All                   │
│ ┌─Current Breakpoints──────────┐  ┌─────────┐    │
│ │ DI 000051c@x sample\\#0047    │  │  Close  │    │
│ │ (EVALUATE *mes)               │  └─────────┘    │
│ │ EN 0000552@x sample\\#0056    │  ┌─────────┐    │
│ │                               │  │  Help   │    │
│ │                               │  └─────────┘    │
│ │ ┌────────────┐ ┌────────┐ ┌───────────┐         │
│ │ │Enable/Disable│ │ Delete │ │ Delete All │     │
│ │ └────────────┘ └────────┘ └───────────┘         │
│ └──────────────────────────────┘                 │
└──────────────────────────────────────────────────┘
```
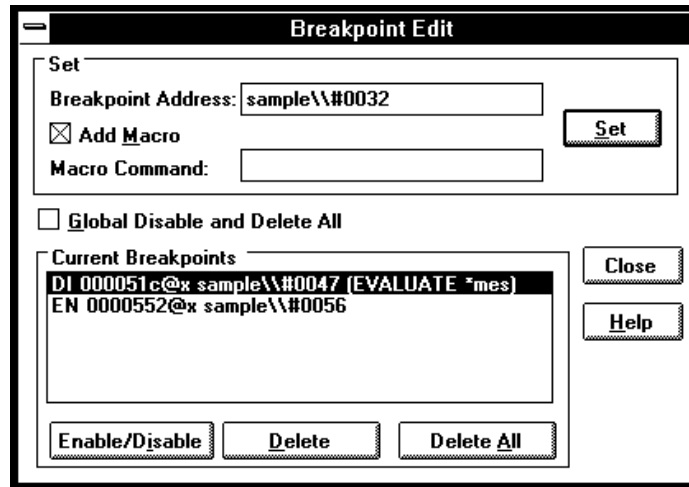
Breakpoint Address	Lets you specify the address at which to set a breakpoint or a break macro.
Add Macro	When selected, this specifies that a break macro should be included with the breakpoint.
Macro Command	Lets you specify the macro to be included with the breakpoint.
Set	Sets a breakpoint with or without a break macro at the specified address.

Global Disable and Delete All	When selected, all existing breakpoints are deleted (not simply disabled), and no new breakpoints can be added.
Current Breakpoints	Displays the addresses and line numbers of the current breakpoints and break macros. Allows you to select the breakpoints or break macros to be enabled/disabled or deleted.
Enable/Disable	Disables or enables the selected breakpoints or breakpoint macros in the Current Breakpoints list box.
	Enabled breakpoints begin with EN in the Current Breakpoints list and show "BP" at the start of the line in the Source window list.
	Disabled breakpoints begin with DI in the Current Breakpoints list and show "bp" at the start of the line in the Source window list.
Delete	Deletes the selected breakpoints or break macros from the Current Breakpoints list box.
Delete All	Deletes all the breakpoints and break macros from the Current Breakpoints list box.
Close	Closes the dialog box.

Command File Command

```
MOD(E) BKP(TBREAK) ON|OFF

BP DEL(ETE) ALL

BP DIS(ABLE) address

BP ENA(BLE) address
```

See Also

"To disable a breakpoint" and
"To list the breakpoints and break macros" in the "Using Breakpoints and Break Macros" section of the "Debugging Programs" chapter.

Variable→Edit... (ALT, V, E)

Displays or modifies the contents of the specified variable or copies it to the WatchPoint window.

A dynamic variable can be registered as a watchpoint when the current program counter is in the function in which the variable is declared. If the program counter is not in this function, the variable name is invalid and an error results.

Variable Edit Dialog Box

Choosing the Variable→Edit... (ALT, V, E) command opens the following dialog box:
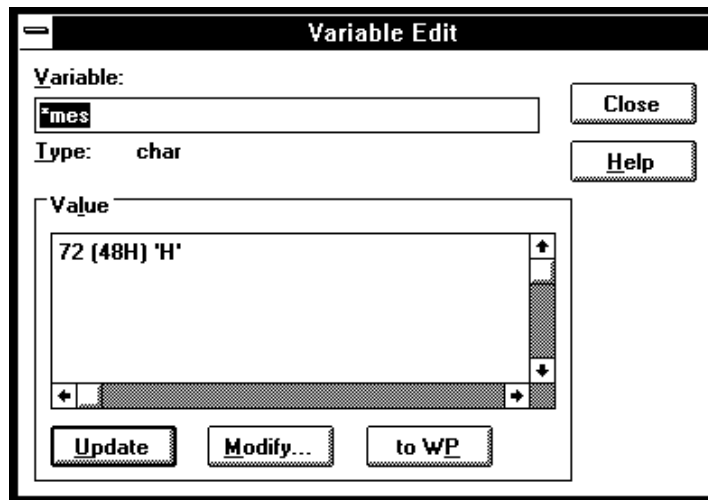
Variable   Specifies the name of the variable to be displayed or modified. The contents of the clipboard, usually a variable selected from the another window, automatically appears in this text box.

Type   Displays the type of the specified variable.

Value   Displays the contents of the specified variable.

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

Command File Command

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

See Also

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

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

Variable Modify Dialog Box

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

Variable	Shows the variable to be edited.	
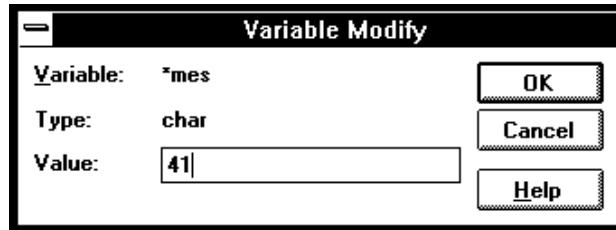
Variable   Shows the variable to be edited.

Type   Indicates the type of the variable displayed in the Variable
                  field.

Value   Lets you enter the new value of the variable.

OK   Replaces the contents of the specified variable with the
                  specified value and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

See Also

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

Trace→Function Flow (ALT, T, F)   Emulator Only

Traces function flow by storing function entry points in the trace buffer.

The analyzer identifies function entry points by looking for a program fetch of the LINK instruction following a pipeline flush.

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

Note

When using the MCC68K compiler, you must specify the -Kf option when compiling programs in order for the debugger to be able to trace function flow. (The -Kf option creates frame pointers for functions.)

Command File Command

```
TRA(CE) FUN(CTION) FLO(W)
```

See Also

"To trace function flow" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Function Caller... (ALT, T, C)   Emulator Only

Traces the caller of the specified function.

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

The analyzer stores only the execution of the function entry point and prestores execution states that occur before the function entry point. These prestored states correspond to the function call statements and identify the caller of the function.

When assembly language programs are used, you can specify the assembler symbol for a subroutine instead of a C function name, and the prestored states will show the instructions that called the subroutine.

Note   Because of prefetching by the 68360 processor, the analyzer may fail in tracing the caller.

Function Caller Trace Dialog Box

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

```
┌─────────────────────────────────────────────┐
│ ▬       Function Caller Trace                │
│ Function:                      ┌──────────┐  │
│                                │    OK    │  │
│ ┌───────────────────────────┐  └──────────┘  │
│ │next_message               │  ┌──────────┐  │
│ └───────────────────────────┘  │  Cancel  │  │
│                                └──────────┘  │
│                                ┌──────────┐  │
│                                │   Help   │  │
│                                └──────────┘  │
└─────────────────────────────────────────────┘
```
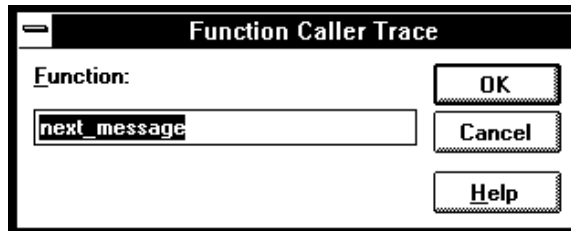
Function   Lets you enter the function whose callers you want to trace.

OK   Executes the command and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) FUNC(TION) CAL(L) address
```

See Also

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

Trace→Function Statement... (ALT, T, S)   Emulator Only

Traces execution within the specified function.

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

The analyzer stores execution states in the function's address range.

Because the analyzer is set up based on function information from the object file, this command cannot be used to trace non-C functions.

Note

The analyzer traces unexecuted instructions due to prefetching by 68360 processor.

Function Statement Trace Dialog Box

Choosing the Trace→Function Statement... (ALT, T, S) command opens the following dialog box:
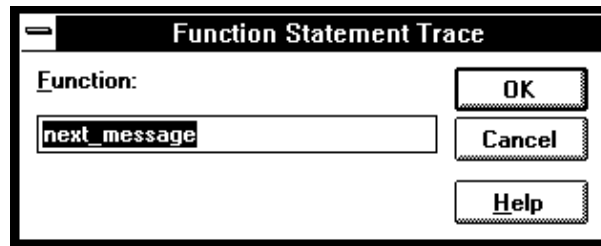
Function   Lets you enter the function whose execution you want to trace.

OK   Traces within the specified function and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

Command File Command

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

See Also

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

Trace→Variable Access... (ALT, T, V)   Emulator Only

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 Access Trace                │
│ Variable:                      ┌──────────┐   │
│                                │    OK    │   │
│ ┌─────────────────────────┐    └──────────┘   │
│ │message_id               │    ┌──────────┐   │
│ └─────────────────────────┘    │  Cancel  │   │
│                                └──────────┘   │
│                                ┌──────────┐   │
│                                │   Help   │   │
│                                └──────────┘   │
└─────────────────────────────────────────────┘
```
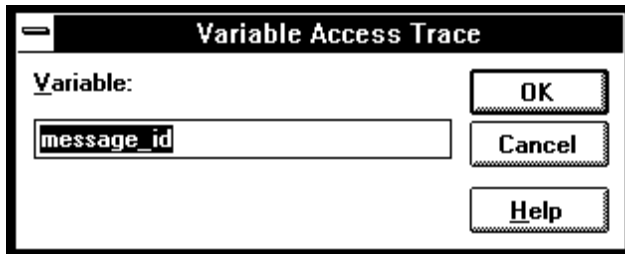
Variable   Lets you enter the variable name.

OK   Traces accesses to the specified variable and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

Command File Command

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

See Also

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

Trace→Variable Break... (ALT, T, B)   Emulator Only

Traces before, and breaks program execution when, a value is written to a variable.

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

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

Variable Break Dialog Box

Choosing the Trace→Variable Break... (ALT, T, B) command opens the following dialog box:
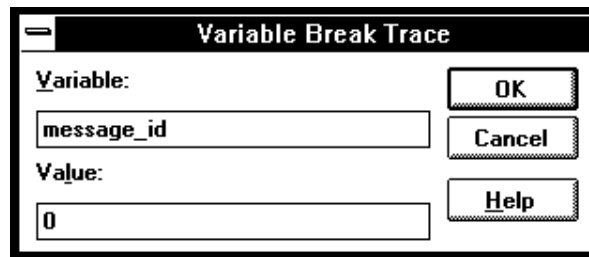
Variable   Lets you enter the variable name.

Value   Lets you enter the value that, when written to the variable, triggers the analyzer.

OK   Starts the trace and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

Command File Command

```
TRA(CE) VAR(IABLE) BRE(AK) address data
```

See Also

"To trace before a particular variable value and break" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Trace→Edit... (ALT, T, E)   Emulator Only

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 the 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)   Emulator Only

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

You can enter address, data, and status values that qualify the state(s) that, when captured by the analyzer, will be stored in the trace buffer or will trigger the analyzer.

Data values are 32-bit values (because the data bus is 32 bits wide). To identify byte or word values on the data bus, use "don't cares" as shown below:

> 1234xxxx
>
> 0xxxx5678
>
> 0xxxx56xx
>
> 0xxxxxx78

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

Note

The analyzer traces unexecuted instructions due to prefetching by the 68360 processor.

Trigger Store Trace Dialog Box

Choosing the Trace→Trigger Store... (ALT, T, T) command opens the
following dialog box:
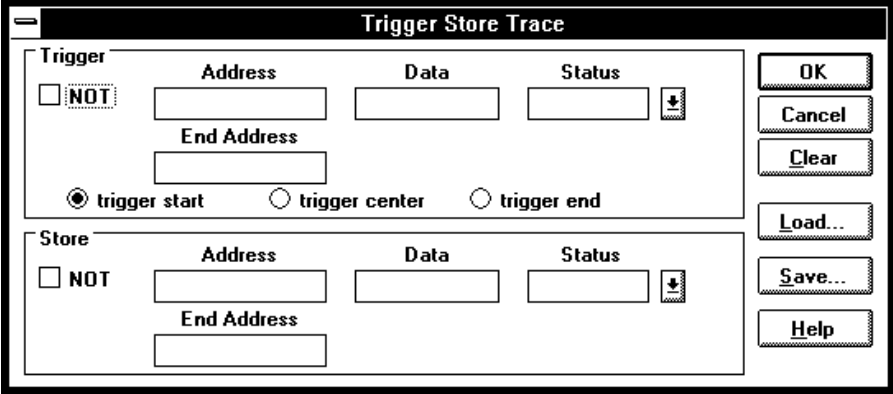
Trigger	This box groups the items that make up the trigger condition.
NOT	Specifies any state that does not match the Address, Data, and Status values.
Address	Specifies the address portion of the state qualifier.
End Address	Specifies the end address of an address range.
Data	Specifies the data portion 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.

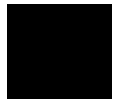## Trace→Find Then Trigger... (ALT, T, D)   Emulator Only

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.

A trace is complete when the trace buffer is full.

Note   The analyzer traces unexecuted instructions due to prefetching by the 68360 processor.

291

Find Then Trigger Trace Dialog Box

Choosing the Trace→Find Then Trigger... (ALT, T, D) command opens the following dialog box:
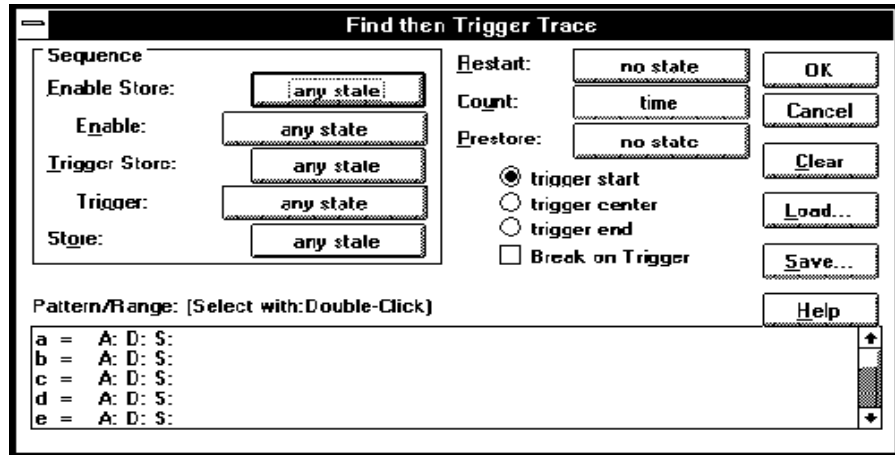
The Sequence group box specifies a two-term sequential trigger condition. It also lets you specify store conditions during the sequence.

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 the 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   When selected, this option specifies that execution break
Trigger   into the monitor when the analyzer is triggered.

Pattern/Range   Specifies the trace patterns for the state conditions.
                     Double-clicking the desired pattern or range 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 in which you specify a name to identify a file containing the present trace specification.

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)   Emulator Only

Traces program execution as specified in the Sequence Trace dialog box.

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

1   Once the trace starts, the analyzer stays on sequence level 1 until the primary or secondary branch condition is found. (If a state satisfies both primary and secondary branch conditions, the primary branch is taken.) Once the primary or secondary branch condition is found, the analyzer transfers to the sequence level specified by the "to" button.

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

3   When the analyzer reaches the sequence level specified in Trigger On, the analyzer is triggered.

4   During the above described operation, the analyzer stores the states specified in the Store text box.

The trace is complete when the trace buffer is full.

Note   The analyzer traces unexecuted instructions due to prefetching by the 68360 processor.

Sequence Trace Dialog Box

Choosing the Trace→Sequence... (ALT, T, Q) command opens the following dialog box:
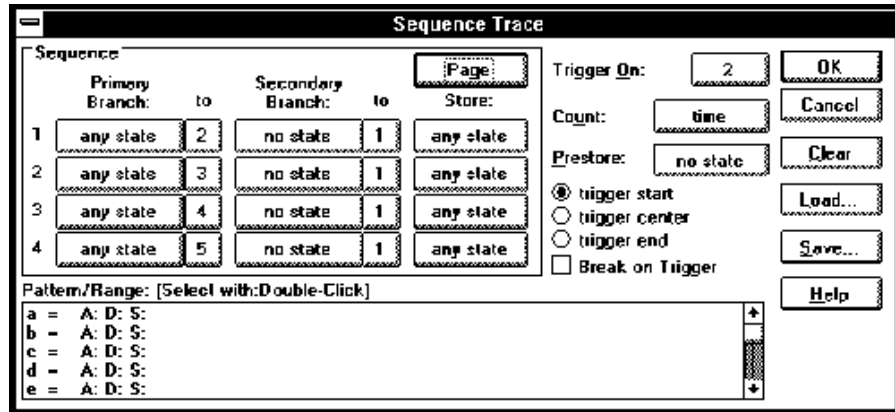
The Sequence group box specifies primary and secondary branch conditions for transferring from one sequence level to another. It also specifies store conditions for each of the eight sequence levels.

Primary Branch   Specifies the condition for transferring to the sequence level specified in the "to" text box.

Secondary   Specifies the condition for transferring to the sequence
Branch   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 to be 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 the 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   When selected, this option specifies that execution break
Trigger   into the monitor when the analyzer is triggered.

Pattern/Range   Specifies the trace patterns for the state conditions.
                 Double-clicking the desired pattern or range 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)   Emulator Only

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.
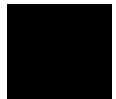
Trace→Halt (ALT, T, H) Emulator Only

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)   Emulator Only

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.
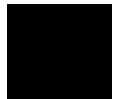
Condition Dialog Boxes   Emulator Only

Choosing the buttons associated with enable, trigger, primary branch, secondary branch, store, or prestore conditions opens the following dialog box:
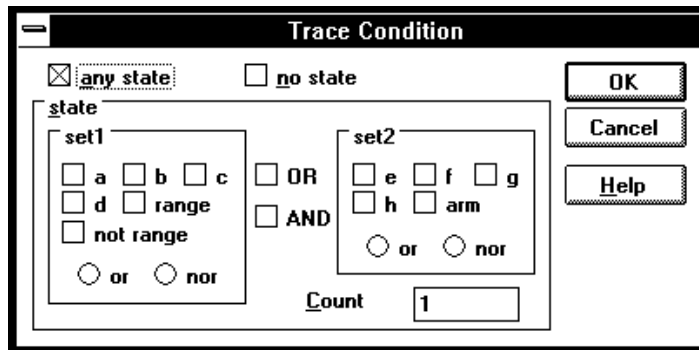
Choosing the button associated with the count condition opens the following dialog box:
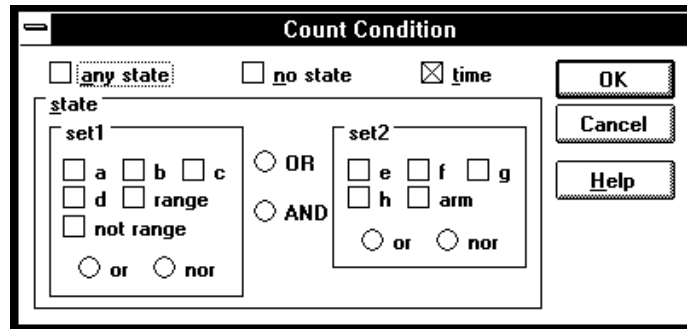
no state	No state meets the specified condition.
any state	Any state meets the specified condition.
time	The analyzer counts time for each state stored in the trace.

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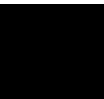
Count	Appearing in Trace Condition dialog boxes, this value specifies the number of occurrences of the state that will satisfy the condition.
OK	Applies the state qualifier to the specified condition and closes the dialog box.
Cancel	Closes the dialog box.

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   Emulator Only

Selecting one of the patterns in the Pattern/Range list box opens the
following dialog box:
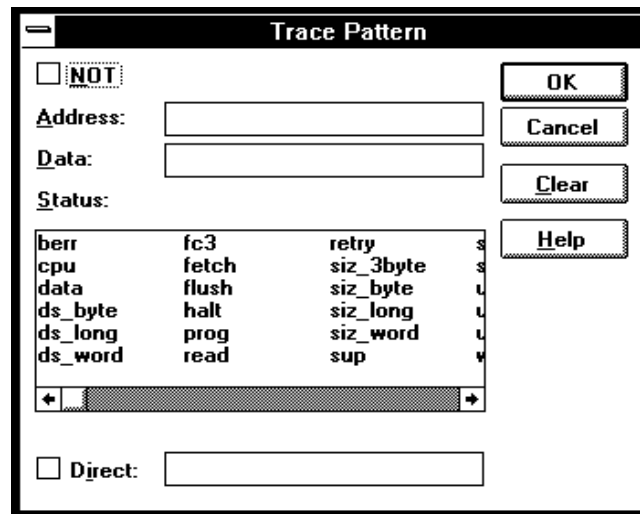
NOT   Lets you specify all values other than the address, data,
                   and/or status values specified.

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

Emulator Only

Selecting the range at the bottom of the Pattern/Range list box opens the following dialog box:
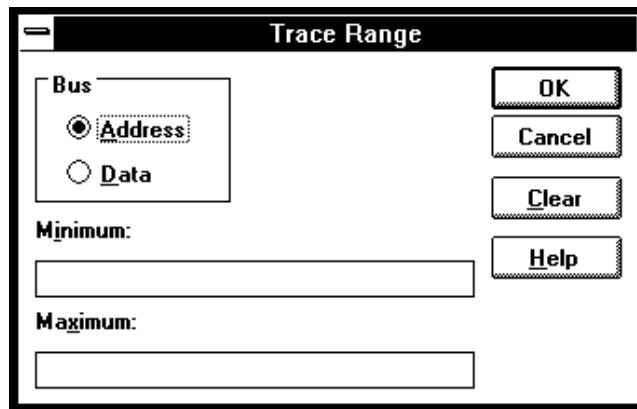
Address	Selects a range of address values.
Data	Selects a range of data values.
Minimum	Lets you enter the minimum value for the range.
Maximum	Lets you enter the maximum value for the range.
OK	Applies the values specified for the range, and closes the dialog box.
Cancel	Closes the dialog box.
Clear	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

Emulator Only

Choosing the buttons associated with "to" or Trigger On opens the following
dialog box:
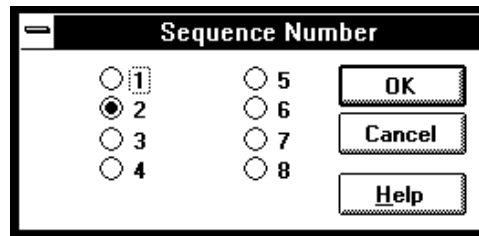
1-8   These options specify the sequence level.

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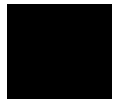
RealTime→I/O Polling→ON (ALT, R, I, O)

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.
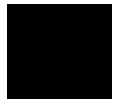
Assemble... (ALT, A)

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:
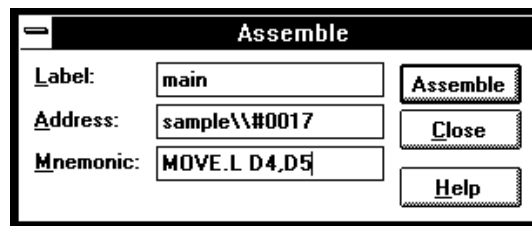
Label   Lets you assign a user-defined symbol to the specified
               address.

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 or HP E3490A Software Probe configuration.

Hardware Config Dialog Box for Emulators

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

Buffering of AS, DS, R/W	Buffering these signals improves signal quality and timing (see the emulator timing specifications that come with the emulator).		
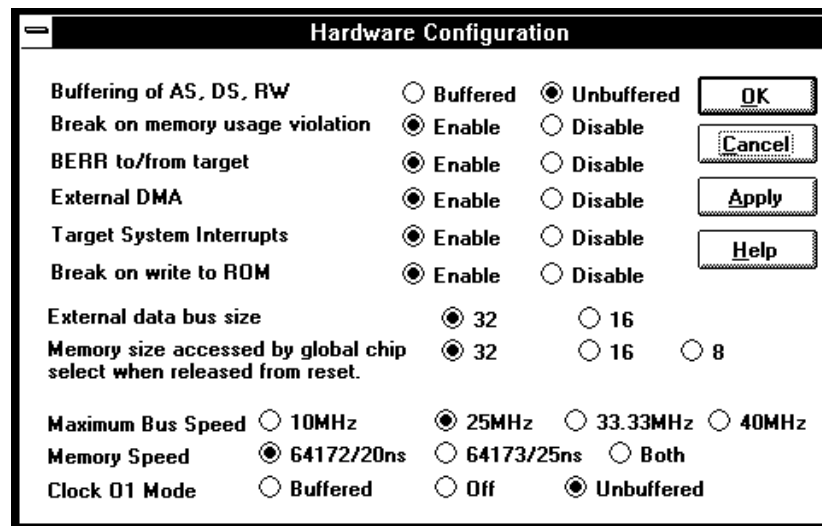
Buffering of AS, DS, R/W   Buffering these signals improves signal quality and timing (see the emulator timing specifications that come with the emulator).

Break on memory usage violation   When enabled, any DRAM/Parity access into emulation memory causes a break. When disabled, these accesses do not cause breaks.

BERR to/from target   When enabled, the connection is complete for driving internal M68360 BERR signals to the target, and for delivering target BERR activity to the M68360. When disabled, internal M68360 BERR signals will not be driven

to the target, and the M68360 will not respond to target BERR activity.

External DMA	When enabled, the emulator supports external DMA to the M68360 and supports DMA tracing, but does not support DMA accesses to emulation memory. When disabled, the emulator does not support any external DMA activity. These options do not affect internal DMA.
Target System Interrupts	When enabled, the emulator responds to interrupts generated by the target system during foreground operation. All interrupts, including level 7 NMI, are blocked when execution is within the background monitor. When disabled, the emulator ignores all interrupts generated by the target system, including level 7 NMI.
Break on write to ROM	Enables or disables breaks to the monitor when the user program writes to memory mapped as ROM.
External data bus size	Specifies the width in bits.
Memory size accessed by global chip select when released from reset	Specifies the size in bits.
Maximum Bus Speed	When 10MHz, the emulator does not add a wait state. You may program chip selects of the M68360 for any TCYC. When 25MHz, the emulator does not add a wait state. Chip selects of the M68360 must be programmed for TCYC >= 1. When 33.33MHz: When the memory type is HP 64172, the emulator will not add a wait state. Chip selects of the M68360 must be programmed for any TCYC >= 1. When the memory type is HP 64173 or both HP 64172 and HP 64173, the emulator will add one wait state. You may program the chip selects of the M68360 for TCYC >= 2.

When 40MHz, the emulator adds one wait state. Chip selects of the M68360 must be programmed for TCYC >= 2.

Memory Speed   Identifies the type of memory modules installed on the emulator probe. This value is used in the wait state calculation associated with Maximum Bus Speed.

Clock O1 Mode   Buffering this output signal improves signal quality. Off specifies that CLKO1 is not driven to the target system.

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) AS_(DS_RW) BUF(FERED)/UNB(UFFERED)

CON(FIG) BRK(MEMVIOL) ENA(BLE)/DIS(ABLE)

CON(FIG) BER(R) ENA(BLE)/DIS(ABLE)

CON(FIG) EXT(DMA) ENA(BLE)/DIS(ABLE)

CON(FIG) TGT(INTR) ENA(BLE)/DIS(ABLE)

CON(FIG) DAT(ABUSSIZE) 16/32

CON(FIG) GCH(IPDSIZE) 8/16/32

CON(FIG) MAX(BUSSPEED) 10MHz/25MHz/33MHz/40MHz

CON(FIG) MEM(SPEED) 20NS/25NS

CON(FIG) CLK(O1) BUF(FERED)/UNB(UFFERED)/OFF
```

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.

Hardware Config Dialog Box for the HP E3490A Software Probe

Choosing the Settings→Emulator Config→Hardware... (ALT, S, E, H)
command opens the following dialog box:
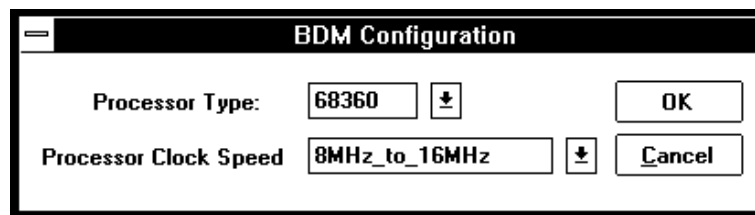
Processor Type   The processor type determines which registers are
                  available in the Real-Time C Debugger.

Processor Clock   The processor clock rate determines the maximum rate
Rate   that the HP E3490A Software Probe can communicate with
                  the target processor through the BDM port.

See Also

The *HP E3490A Software Probe User's Guide.*
"Setting the Hardware Options" in the "Configuring the Emulator" chapter.

Command File Command

CFGBDM SPEED 4/8/16/20

CFGBDM TYPE 68360

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).

The emulator provides two slots for emulation memory modules. The amount of emulation memory that can be mapped depends on the number, and size, of memory modules installed.

Emulation memory is made available to the mapper in blocks. When you map an address range to emulation memory, at least one block is assigned to the range. When a block of emulation memory is assigned to a range, it is no longer available, even though part of the block may not be used. Emulation memory in either slot of the emulator probe is divided into four equal blocks.

When you map ranges of emulation memory, blocks are allocated so as to leave the greatest amount of emulation memory available.

Memory Map Dialog Box

Choosing the Settings→Emulator Config→Memory Map... (ALT, S, E, M) command opens the following dialog box:
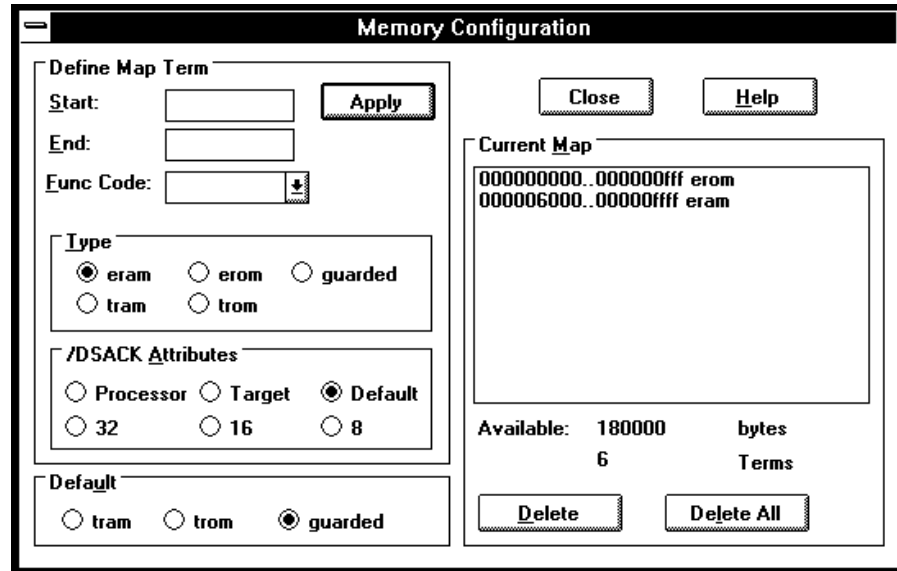
Start   Specifies the starting address of the address range to be mapped.

End   Specifies the end address of the address range to be mapped.

Func Code   Assigns any of the *function codes* to the address range. It is only necessary to specify a function code other than X when mapping overlapping address ranges for different memory spaces. When mapping overlapping ranges, you can only select function codes that haven't already been selected for previously mapped ranges.

Type   Lets you select the memory type of the specified address range. 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.
DSACK Attributes	These options specify where /DSACK signals come from for accesses in the range.
	Processor = /DSACKs are from the processor. Target = /DSACKs are from the target system. Default = 32-bit /DSACKs are from the emulator. 32 = 32-bit /DSACKs are from the emulator. 16 = 16-bit /DSACKs are from the emulator. 8 = 8-bit /DSACKs are from the emulator.
Apply	Maps the address range specified in the Define Map Term group box.
Default	Specifies whether unmapped memory ranges are target system RAM, target system ROM, or guarded memory.
Current Map	Lists currently mapped ranges.
Available	Indicates the amount of emulation memory available.
Delete	Deletes the address range selected in the Current Map list box.
Delete All	Deletes all of the address ranges in the Current Map list box.
Close	Closes the dialog box.

Command File Command

```
MAP addressrange mem_type func_code attribute
```
Maps the specified address range with the specified memory type and
function code. The /DSACK attributes can be: dsup, dstgt, ds32, ds16, or ds8.
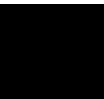
`MAP OTH(ER) mem_type`
Specifies the type of the specified non-mapped memory area.

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)

Emulator Only

Selects the type of monitor program and other monitor options.

Target system interrupts are blocked during background monitor operation, but they may be enabled during foreground monitor operation.

Monitor Config Dialog Box

Choosing the Settings→Emulator Config→Monitor... (ALT, S, E, O) command opens the following dialog box:
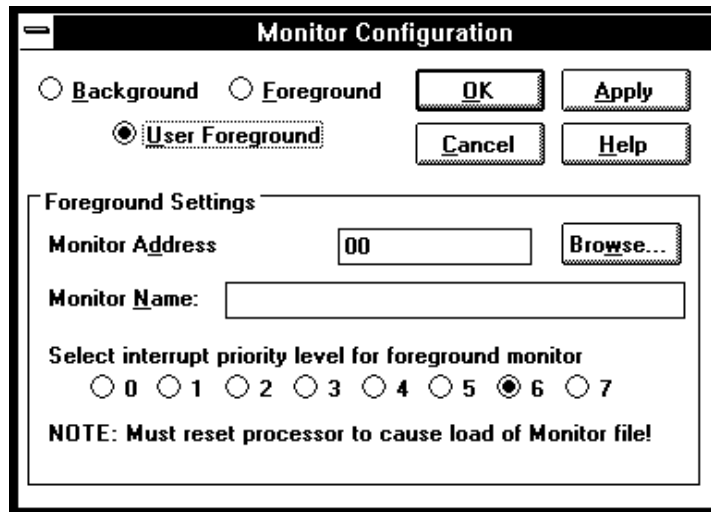
Background/ Foreground/ User Foreground	Select between the background monitor (which is implemented with the 68360 Background Debug Mode (BDM)), the foreground monitor (which runs out of the same address space as your programs), or the user foreground monitor (which is your customized version of the foreground monitor).
Monitor Address	When a foreground monitor is selected, this is the 4-Kbyte boundary address (ending in 000h) that is the base of the 4-Kbyte block of memory used by the monitor. Four

	Kbytes of emulation memory must be mapped at this address.
Monitor Name	Lets you enter the name of the user foreground monitor object file. The default is C:\HP\RTC\M360\FGMON\FGMON.X (if C:\HP\RTC\M360 was the installation path chosen when installing the debugger software).
Browse...	Opens a file selection dialog box from which you can select the foreground monitor object file to be loaded.
Select interrupt priority level for foreground monitor	This option lets you run the foreground monitor at a lowered interrupt priority level in order to allow critical target system interrupts to be processed.
	When it's safe to lower the interrupt level, the foreground monitor will set the interrupt priority mask to either the level entered or the level in effect before monitor entry, whichever is greater.
OK	Modifies the monitor configuration as specified and closes the dialog box.
	If you have selected a foreground monitor, it is loaded automatically after each Emulation→Reset (ALT, E, E) command.
Cancel	Cancels the monitor configuration and closes the dialog box.
Apply	Same as the OK button, except the dialog is left open.

Command File Command

```
MON(ITOR) TYP(E) BAC(KGROUND)

MON(ITOR) TYP(E) FOR(EGROUND)

MON(ITOR) TYP(E) USE(RFOREGROUND)

MON(ITOR) ADD(RESS) address
```

```
MON(ITOR) FIL(ENAME) file_name
```

```
MON(ITOR) INT(ERRUPT) pri_level
```

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→Information... (ALT, S, E, I)

This command lets you:

- Check the emulator configuration for inconsistencies.

- Display decoded and formatted information about the emulator configuration.

- Synchronize the 68360 system integration module (SIM) registers to the emulator's EMSIM registers.

Configuration Information Dialog Box

Two list boxes let you select the operation. Each has a button that confirms the selection. The results are displayed in the viewing area.
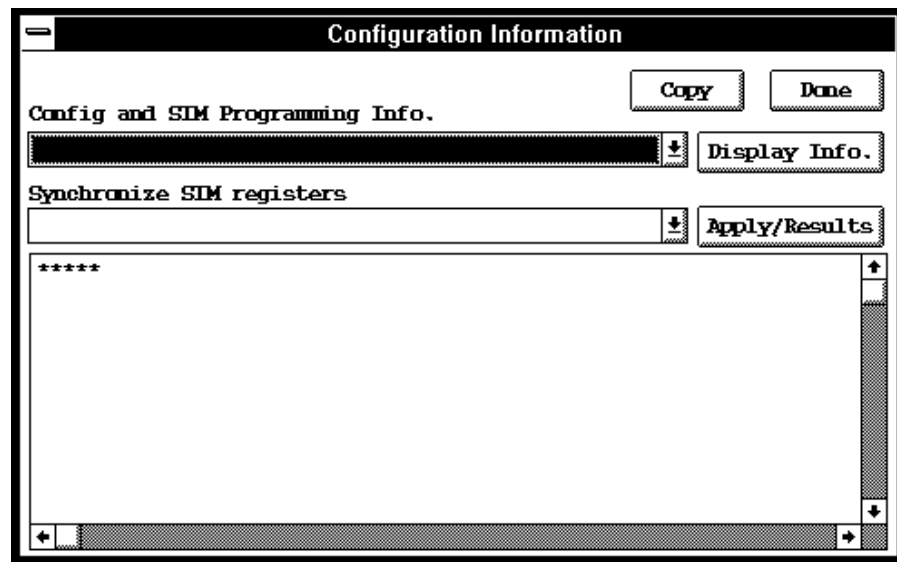
Config and SIM   You can select:
Programming
Info.   Check emulator configuration which displays error
                    messages that result from inconsistencies between related
                    configuration values. These errors should be resolved for
                    the emulator to operate correctly. In addition, status
                    messages about expectations and limitations of the
                    emulator are displayed.

                    Chip selects in SIM (processor) register set which shows
                    how chip selects are defined.

                    Chip selects in EMSIM (emulator) register set which shows
                    how chip selects are defined.

                    Bus interface ports in SIM (processor) register set which
                    shows how bus interface ports are defined.

                    Bus interface ports in EMSIM (emulator) register set which
                    shows how bus interface ports are defined.

                    Memory map & correlation with CSs, IM reg blk & RAM
                    which shows the memory map and its correlation with chip
                    selects, internal module register block, and RAM.

                    Reset mode configuration value and operation which shows
                    how the reset mode configuration is defined.

                    Upper address mode of emulator for 68360 which shows
                    how the upper address mode is defined.

                    Clock input mode which shows how the clock input mode
                    is defined.

                    Assembly listing matching current EMSIM registers which
                    shows the assembly language code that would initialize the
                    processor to be the same as the current EMSIM register set.

Display Info.   Performs the selected operation and displays the results in
                    the viewing area.

Synchronize SIM registers	You can select:
	Synchronize from '360 sim regs, copy to emsim regs which programs the emulator's EMSIM registers from the 68360 SIM. This is useful if initialization code that configures the 68360 SIM exists, but you don't know what its values are. In this case, you can use the default configuration, run from reset to execute the initialization code, and synchronize the EMSIM registers to match the 68360 SIM.
	Synchronize from emsim regs, copy to '360 registers which transfers the programming of the EMSIM registers into the 68360 SIM. This happens automatically each time a break to the monitor from emulation reset occurs; this ensures that the 68360 is prepared to properly access memory when a program is downloaded to the emulator.
	Show differences for M68360 and emsim registers.
	Default the emsim register set which resets the EMSIM registers to default processor values.
Apply/Results	Performs the selected operation and displays the results in the viewing area.
Copy	Opens a file selection dialog box that lets you select the file to which information in the viewing area is copied.
Done	Closes the dialog box.

Command File Command

```
CFG(INFO) INF(O) 0
```
Check emulator configuration.

```
CFG(INFO) INF(O) 1
```
Chip selects in SIM (processor) register set.

```
CFG(INFO) INF(O) 2
```
Chip selects in EMSIM (emulator) register set.

```
CFG(INFO) INF(O) 3
```
Bus interface ports in SIM (processor) register set.

```
CFG(INFO) INF(O) 4
```
Bus interface ports in EMSIM (emulator) register set.

```
CFG(INFO) INF(O) 5
```
Memory map & correlation with CSs, IM reg blk & RAM.

```
CFG(INFO) INF(O) 6
```
Reset mode configuration value and operation.

```
CFG(INFO) INF(O) 7
```
Upper address mode of emulator for 68360.

```
CFG(INFO) INF(O) 8
```
Clock input mode.

```
CFG(INFO) INF(O) 9
```
Assembly listing matching current EMSIM registers.

```
CFG(INFO) SYN(C) 0
```
Synchronize from '360 sim regs, copy to emsim regs.

```
CFG(INFO) SYN(C) 1
```
Synchronize from emsim regs, copy to '360 registers.

```
CFG(INFO) SYN(C) 2
```
Show differences for M68360 and emsim registers.

```
CFG(INFO) SYN(C) 3
```
Default the emsim register set.

See Also

"Using the EMSIM Registers" in the "Configuring the Emulator" chapter.

"Verifying the Emulator Configuration" in the "Configuring the Emulator"
chapter.
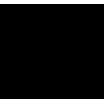
Settings→Communication... (ALT, S, C)

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

RTC Emulation Connection Dialog Box

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

Current Connection Status

> This part of the dialog box shows the current communication settings.

RTC Core Version Information

> Displays software version information.

New Emulator Connection Setup

Transport   Lets you choose the type of connection to be made to the
Selection   HP 64700. Double-clicking causes the current connection
                   to be tried with the given transport. Single-clicking selects
                   the transport for use with the Setup button.

User Name   This name tells the HP 64700 and other users who you are.
                   When other users attempt to access the HP 64700 while
                   you are using it or while it is locked, a message tells them
                   you're using it.

User ID   Another method of identifying yourself to the HP 64700
                   and other users. This is primarily useful in a mixed UNIX
                   and MS-DOS environment; when a UNIX user tries to
                   unlock an emulator, the user ID is used to look into the
                   /etc/passwd entry on the UNIX host for the user name.

                   If your HP 64700 is on the LAN, we recommend that you
                   change User Name and User ID so that other users can
                   easily tell if an emulator is in use and by whom. Also, if you
                   don't change the User Name/ID from the defaults, the
                   File→Exit HW Locked (ALT, F, H) command has no effect
                   because all users are identical.

Setup   Opens a transport-specific dialog box which usually allows
                   you to change the address and unlock the emulator.

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

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

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

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

The Real-Time C Debugger does not allow you to change connection or transport information without leaving the debugger and reentering it. However, any changes you make will be put in the .INI file and take effect the next time you enter the debugger (assuming that you do not override the .INI information on the command line).

The command line options for connection and transport (-E and -T) take precedence over the values in the .lNI file.

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

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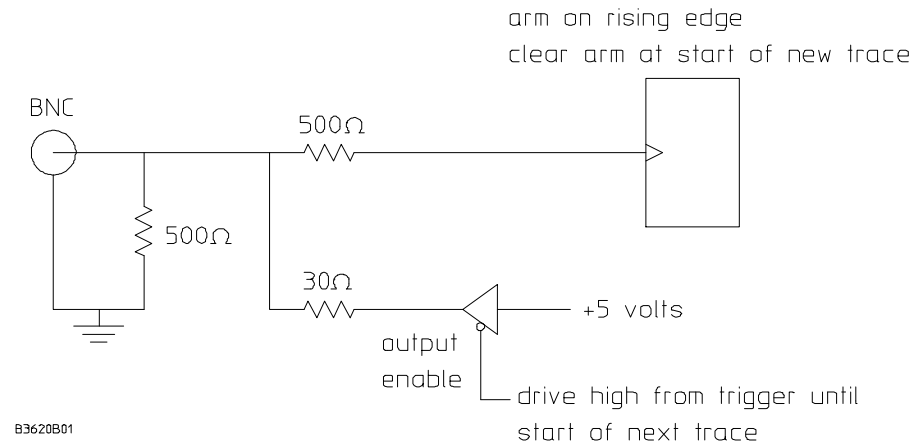
When a trace starts, it stops driving the output (so if nothing else is driving the line, it will fall low due to the 500-ohm pull-down resistor).

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)
Emulator Only

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

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

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:
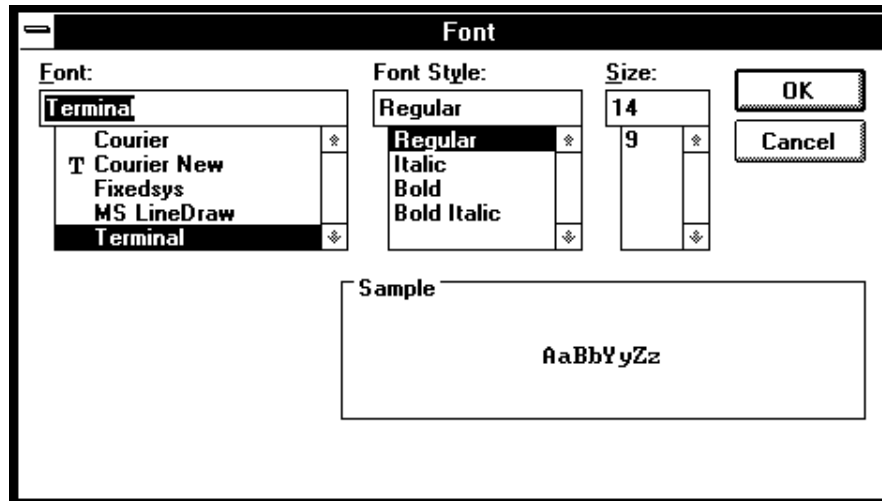
Font	Lets you select the font to be used in the Real-Time C Debugger interface. The "T" shaped icon indicates a TrueType font.
Font Style	Lets you select the typeface, for example, regular, bold, italic, etc.
Size	Lets you select the size of the characters.
Sample	Shows you what the selected font looks 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:
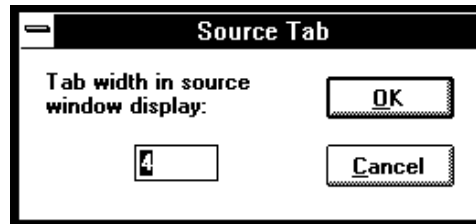
Tab width in source window display — Enter the number of spaces between tab stops. This also affects the tab width for source lines in the Trace window. The number must be between 1 and 20.

OK — Sets the tab stops, and closes the dialog box.

Cancel — Cancels tab stop setting, and closes the dialog box.

See Also

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

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

Symbol database search is case sensitive.

Command File Command

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

See Also

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

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

Symbol database search is not case sensitive.

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

Command File Command

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

See Also

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

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

Emulator Only

Traces foreground emulation microprocessor operation.

This is the normal setting.

Command File Command

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

See Also

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

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

Emulator Only

Traces background emulation microprocessor operation.

This is rarely a useful setting when debugging programs.

Command File Command

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

See Also

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

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

Traces both foreground and background emulation microprocessor operation.

Command File Command

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

See Also

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

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

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

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

Command File Command

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

See Also

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

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

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

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

Command File Command

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

See Also

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

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

You are prompted for source file paths.

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

Command File Command

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

See Also

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

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

You are not prompted for source file paths.

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

Command File Command

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

See Also

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

Window→Cascade (ALT, W, C)

Arranges, sizes, and overlaps windows.

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

Window→Tile (ALT, W, T)

Arranges and sizes windows so that none are overlapped.

Windows are sized evenly.

Window→Arrange Icons (ALT, W, A)

Rearranges icons in the Real-Time C Debugger window.

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

Window→1-9 (ALT, W, 1-9)

Opens the window associated with the number.

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

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

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

Command File Command

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

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

See Also

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

Window→More Windows... (ALT, W, M)

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

Select Window Dialog Box

Choosing the Window→More Windows... (ALT, W, M) command opens the
following dialog box:
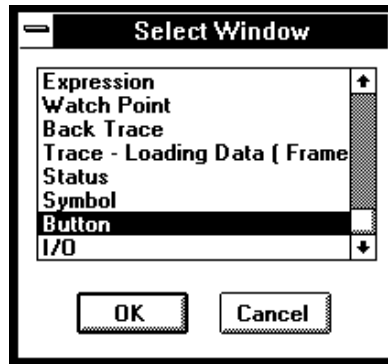
OK   Opens the window selected in the list box.

Cancel   Closes the dialog box.

Command File Command

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

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

See Also

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

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

Provides information on the Real-Time C Debugger.

Choosing the Help→About Debugger/Emulator... (ALT, H, D) command opens a dialog box containing the version information on the current Real-Time C Debugger and emulator.

Source Directory Dialog Box

When the source file associated with a symbol cannot be found in the current directory, the following dialog box is opened:
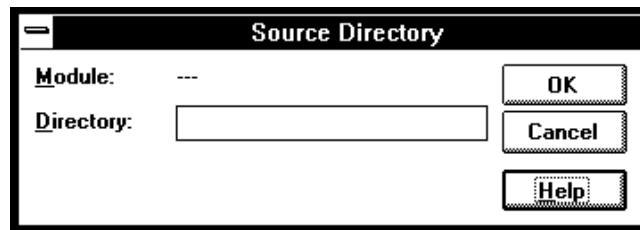
Module   Shows the symbol whose source file could not be found.

Directory   Lets you enter the directory in which the source file associated with the symbol may be found.

OK   Adds the directory entered in the Directory text box to the source file search path.

Cancel   Closes the dialog box.

WAIT Command Dialog Box

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

Choosing the STOP button cancels the WAIT command.

10

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

- Register Window Commands

- Source Window Commands

- Symbol Window Commands

- Trace Window Commands

- WatchPoint Window Commands

Common Control Menu Commands

This section describes commands that appear in the control menus of most of the debugger windows:

- Copy→Window (ALT, -, P, W)

- Copy→Destination... (ALT, -, P, D)

Copy→Window (ALT, -, P, W)

Copies the current window contents to the destination file specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
COP(Y) BAC(KTRACE)

COP(Y) BUT(TON)

COP(Y) EXP(RESSION)

COP(Y) IO

COP(Y) MEM(ORY)

COP(Y) REG(ISTER)

COP(Y) SOU(RCE)

COP(Y) WAT(CHPOINT)
```

See Also

"To copy window contents to the list file" in the "Working with Debugger Windows" section of the "Using the Debugger Interface" chapter.
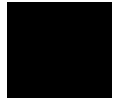
Copy→Destination... (ALT, -, P, D)

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

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

Command File Command

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

See Also

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

Button Window Commands

This section describes the following command:

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

Edit... (ALT, -, E)

Lets you define and label buttons in the Button window.

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

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

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

Button Edit Dialog Box

Choosing the Edit... (ALT, -, E) command opens the following dialog box:
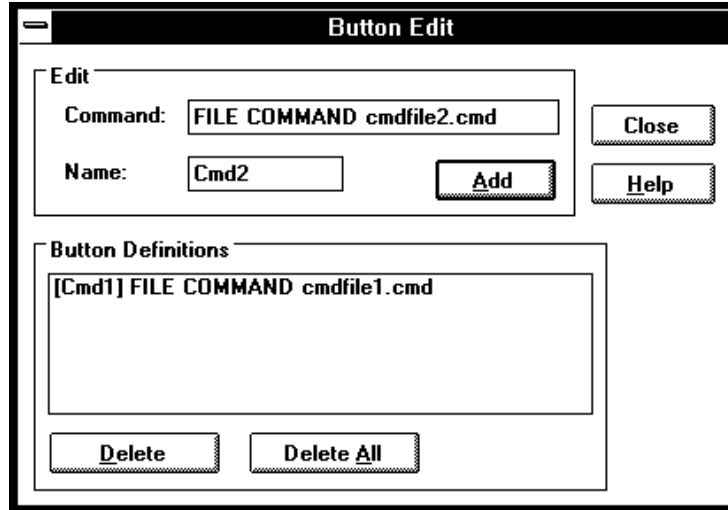
Command   Specifies the command to be associated with the button.
               Command syntax is described at the bottom of most help
               topics under the "Command File Command" heading. Also,
               look in the "Command File and Macro Command Summary"
               chapter in the "Reference" part.

               You can only enter a single command here; if you want a
               series of commands to be executed when this button is
               used, put them in a command file and use the command
               "FILE COMMAND filename," where "filename" is the name
               of your command file.

Name   Specifies the button label to be associated with the
               command.

Add   Adds the button to the button window.

Button   Lists the currently defined buttons. You can select button
Definitions   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.
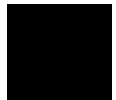
Expression Window Commands

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:

```
┌─────────────────────────────────────────┐
│ ▬        Evaluate Expression             │
│ Expression:                ┌──────────┐  │
│                            │ Evaluate │  │
│ ┌──────────────────────┐   └──────────┘  │
│ │ &message_id          │   ┌──────────┐  │
│ └──────────────────────┘   │  Close   │  │
│                            └──────────┘  │
│                            ┌──────────┐  │
│                            │  Help    │  │
│                            └──────────┘  │
└─────────────────────────────────────────┘
```
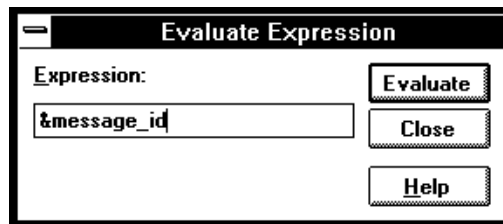
Expression   Lets you enter the expression to be evaluated.

Evaluate   Makes the evaluation and places the results in the
                 Expression window.

Close   Closes the dialog box.

Command File Command

EVA(LUATE) address

EVA(LUATE) "strings"

See Also

"Symbols" in the "Expressions in Commands" chapter.
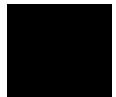
I/O Window Commands

This section describes the following command:

• Define... (ALT, -, D)

Define... (ALT, -, D)

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

I/O Setting Dialog Box

Choosing the Edit→Definition... command opens the following dialog box:
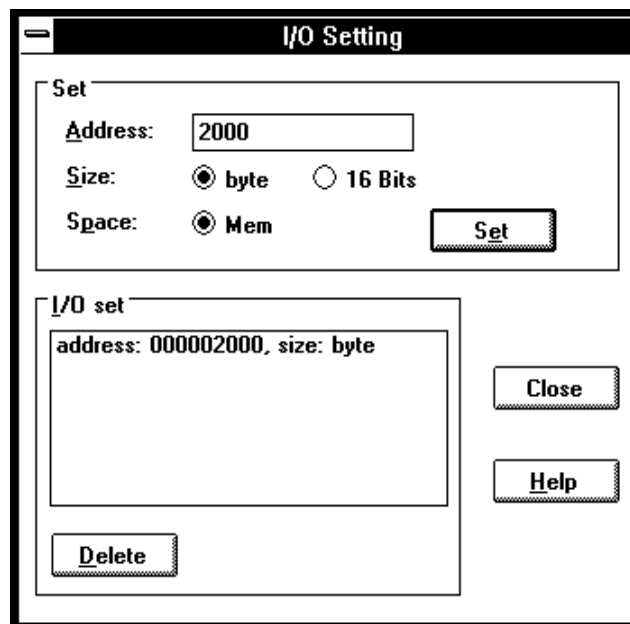
Address   Specifies the address of the I/O location to be defined.

Size   Specifies the data format of the I/O location to be defined.
                   You can select the Byte or 16 Bits option.

Space   Specifies whether the I/O location is in memory or I/O
                   space.

Set   Adds the specified I/O location.

I/O set   Displays the information on the I/O locations that have
                   been set.

Delete   Deletes the I/O locations selected in the I/O set list box.

Close   Closes the dialog box.

Command File Command

```
IO BYTE/WORD/LONG IOSPACE/MEMORY address TO data
```
Replaces the contents of the specified I/O address with the specified value in
the specified size.

```
IO SET BYTE/WORD/LONG IOSPACE/MEMORY address
```
Registers the I/O address to be displayed in the specified size.

```
IO DEL(ETE) BYTE/WORD/LONG IOSPACE/MEMORY address
```
Deletes the I/O specified with its address and size.

See Also

"Displaying and Editing I/O Locations" in the "Debugging Programs" chapter.
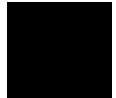
Memory Window Commands

This section describes the following commands:

- Display→Linear (ALT, -, D, L)

- Display→Block (ALT, -, D, B)

- Display→Byte (ALT, -, D, Y)

- Display→16 Bits (ALT, -, D, 1)

- Display→32 Bits (ALT, -, D, 3)

- Search... (ALT, -, R)

- Utilities→Copy... (ALT, -, U, C)

- Utilities→Fill... (ALT, -, U, F)

- Utilities→Load... (ALT, -, U, L)

- Utilities→Store... (ALT, -, U, S)

Display→Linear (ALT, -, D, L)

Displays memory contents in single column format.

Command File Command

MEM(ORY) ABS(OLUTE)

Display→Block (ALT, -, D, B)

Displays memory contents in multi-column format.

Command File Command

MEM(ORY) BLO(CK)

Display→Byte (ALT, -, D, Y)

Displays memory contents as bytes.

Command File Command

MEM(ORY) BYTE

Display→16 Bit (ALT, -, D, 1)

Displays memory contents as 16-bit values.

Command File Command

MEM(ORY) WORD

Display→32 Bit (ALT, -, D, 3)

Displays memory contents as 32-bit values.

Command File Command

MEM(ORY) LONG

Search... (ALT, -, R)

Searches for a value or string in a range of memory.

When the value or string is found, the location is displayed in the Memory window. Choose the Window→Memory command to open the window.

The value or string can be selected from another window (in other words, copied to the clipboard) before choosing the command; the contents of the clipboard will automatically appear in the dialog box that is opened.

Search Memory Dialog Box

Choosing the Search... (ALT, -, R) command opens the following dialog box:
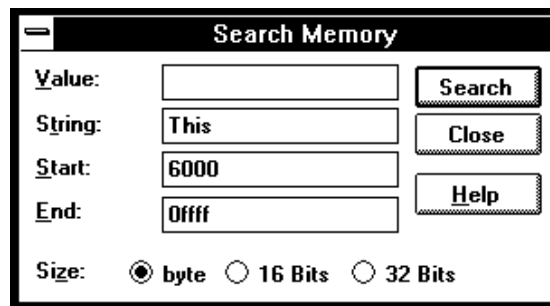
Value	Lets you enter a value.
String	Lets you enter a string.
Start	Lets you enter the starting address of the memory range to search.
End	Lets you enter the end address of the memory range to search.
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.
Execute	Searches for 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:
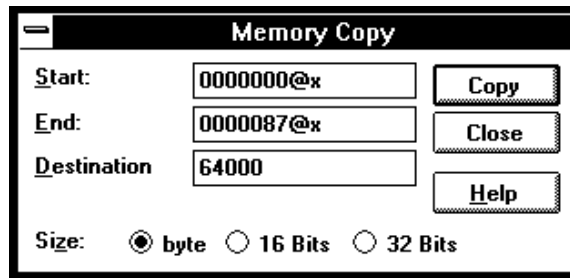
Start	Lets you enter the starting address of the source memory area.	
End	Lets you enter the end address of the source memory area.	
Destination	Specifies the starting address of the destination memory area.	
Size	Selects the data size using the Byte, 16 Bits, or 32 Bits option buttons.	
Execute	Copies 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.	
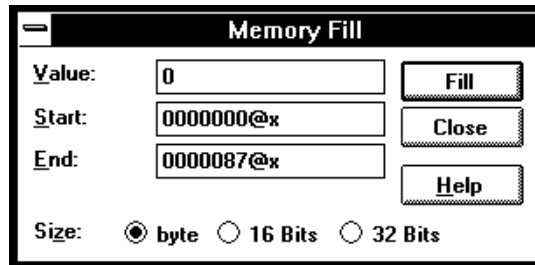
Value   Lets you enter the filling value.

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→Load... (ALT, -, U, L)

Loads memory contents from a previously stored file.

Load Binary File Dialog Box

Choosing the Utilities→Load... (ALT, -, U, L) command opens the following dialog box:
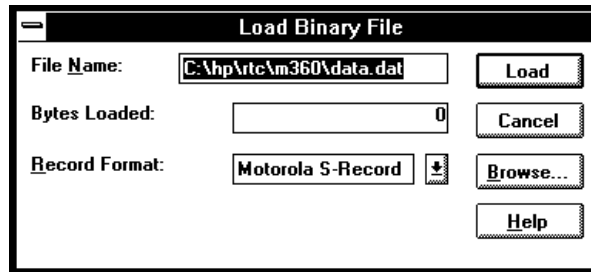
File Name   Lets you enter the name of the file to load memory from.

Bytes Loaded   After you choose the Import button, this box shows the
                 number of bytes that are loaded.

Record Format   Lets you specify the format of the file from which you're
                 loading memory. You can load Motorola S-Record or Intel
                 Hexadecimal format files.

Load   Starts the memory load.

Cancel   Closes the dialog box.

Browse...   Opens a file selection dialog box from which you can select
                  the file name.

Command File Command

```
MEM(ORY) LOA(D) MOT(OSREC) filename
```

```
MEM(ORY) LOA(D) INT(ELHEX) filename
```

See Also

"To copy target system memory into emulation memory" in the "Displaying
and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Store... (ALT, -, U, S)
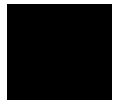
Utilities→Store... (ALT, -, U, S)

Stores memory contents to a binary file.

Store Binary File Dialog Box

Choosing the Utilities→Store... (ALT, -, U, S) command opens the following
dialog box:
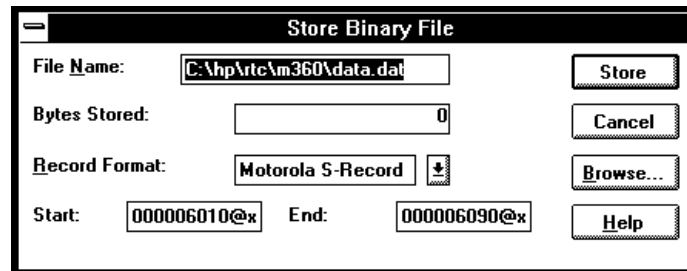
File Name	Lets you enter the name of the file to which memory contents are stored.
Bytes Stored	After you choose the Export button, this box shows the number of bytes that are stored.
Record Format	Lets you specify the format of the file to which you're storing memory. You can select Motorola S-Record or Intel Hexadecimal formats.
Start	Lets you enter the starting address of the memory range to be stored.
End	Lets you enter the ending address of the memory range to be stored.
Store	Starts the memory store.
Cancel	Closes the dialog box.

Browse...   Opens a file selection dialog box from which you can select
                   a file name.

Command File Command

```
MEM(ORY) STO(RE) MOT(OSREC) addr-range filename
```

```
MEM(ORY) STO(RE) INT(ELHEX) addr-range filename
```

See Also

"To copy target system memory into emulation memory" in the "Displaying
and Editing Memory" section of the "Debugging Programs" chapter.

Utilities→Load... (ALT, -, U, L)
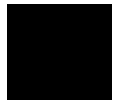
Register Window Commands

This section describes the following command:

- Copy→Registers (ALT, -, P, R)

Copy→Registers (ALT, -, P, R)

Copies the current Register window contents to the destination file specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

```
COP(Y) REG(ISTER)
```

Register Bit Fields Dialog Box

When a register has bit fields, a dialog will pop up and the register value may be edited by changing the whole value or by editing individual bit fields.

When editing in the dialog box, a carriage return is the same as choosing the OK button. To end an edit of a field within the dialog box without quitting, use the Tab key.

The description below a bit field name tells you what has been selected. This description changes when you change the contents of the bit field.

Edited Value   Shows the register value that corresponds to the selections made below. You can also change the register's value by modifying the value in this text box.

Original Value   Shows the value of the register when the dialog box was opened. If the register could not be read, 'XXXXXXXX' is displayed.

OK   Modifies the register as specified, and closes the dialog box.

Cancel   Closes the dialog box without modifying the register.

Source Window Commands

This section describes the following commands:

- Display→Mixed Mode (ALT, -, D, M)

- Display→Source Only (ALT, -, D, S)

- Display→Select Source... (ALT, -, D, L)

- Search→String... (ALT, -, R, S)

- Search→Function... (ALT, -, R, F)

- Search→Address... (ALT, -, R, A)

- Search→Current PC (ALT, -, R, C)

Display→Mixed Mode (ALT, -, D, M)
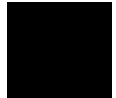
Chooses the source/mnemonic mixed display mode.

Command File Command

```
MOD(E) MNE(MONIC) ON
```

See Also

"To display source code mixed with assembly instructions" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Display→Source Only (ALT, -, D, S)

Chooses the source only display mode.

Command File Command

MOD(E) MNE(MONIC) OFF

See Also

"To display source code only" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Display→Select Source... (ALT, -, D, L)

Displays the contents of the specified C source file in the Source window.

This command is disabled before the object file is loaded or when no source is available for the loaded object file.

Select Source Dialog Box

Choosing the Display→Select Source... (ALT, -, D, L) command opens the following dialog box:
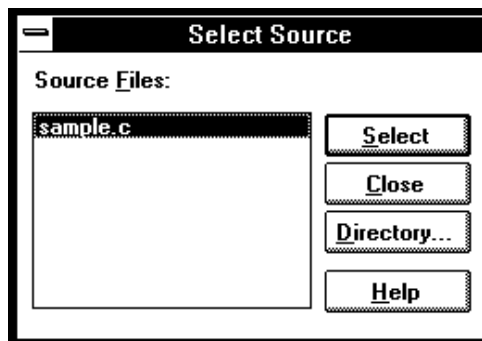
Source Files   Lists C source files associated with the loaded object file. You can select the source file to be displayed from this list.

Select   Switches the Source window contents to the selected source file.

Close   Closes the dialog box.

Directory   Opens the Search Directories Dialog Box from which you can add directories to the search path.

Command File Command

FIL(E) SOU(RCE) module_name

See Also

"To display source files by their names" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Search→String... (ALT, -, R, S)

Searches for, and displays, a string in the Source window.

The search starts from the current cursor position in the Source window, may be either forward or backward, and may be case sensitive.

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

Search String Dialog Box

Choosing the Search→String... (ALT, -, R, S) command opens the following dialog box:
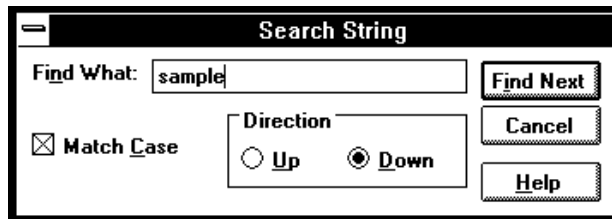
Find What   Lets you enter the string.

Match Case   Selects or deselects case matching.

Up   Specifies that the search be from the current cursor position backward.

Down   Specifies that the search be from the current cursor position forward.

Find Next   Searches for the string.

Close   Closes the dialog box.

Command File Command

`SEA(RCH) STR(ING) FOR/BACK ON/OFF strings`
Searches the specified string in the specified direction with the case
matching option ON or OFF.

See Also

"To search for strings in the source files" in the "Loading and Displaying
Programs" section of the "Debugging Programs" chapter.

Search→Function... (ALT, -, R, F)

Searches for, and displays, a function in the Source window.

The object file and symbols must be loaded before you can choose this
command.

Note   This command displays the source file based on the function information in
the object file. Depending on the structure of the function, the command
may fail in displaying the declaration of the function.

Search Function Dialog Box

Choosing the Search→Function... (ALT, -, R, F) command opens the
following dialog box:
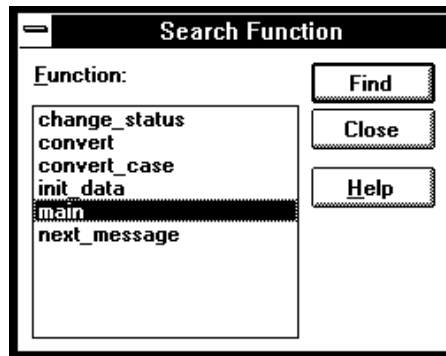
Function   Lets you select the function to search for.

Find   Searches the specified function.

Close   Closes the dialog box.

Command File Command

```
SEA(RCH) FUNC(TION) func_name
```

See Also

"To search for function names in the source files" in the "Loading and
Displaying Programs" section of the "Debugging Programs" chapter.

Search→Address... (ALT, -, R, A)

Searches for, and displays, an address in the Source window.

Address expressions such as function names or symbols can be selected from another window (in other words, copied to the clipboard) before choosing the command; the contents of the clipboard will automatically appear in the dialog box that is opened.

Search Address Dialog Box

Choosing the Search→Address... (ALT, -, R, A) command opens the following dialog box:

```
┌──────────────────────────────────────────┐
│ ▬           Search Address                │
├──────────────────────────────────────────┤
│ Address:                   ┌──────────┐   │
│                            │   Find   │   │
│ ┌────────────────────┐     └──────────┘   │
│ │ 6a6                │     ┌──────────┐   │
│ └────────────────────┘     │  Close   │   │
│                            └──────────┘   │
│                            ┌──────────┐   │
│                            │   Help   │   │
│                            └──────────┘   │
└──────────────────────────────────────────┘
```
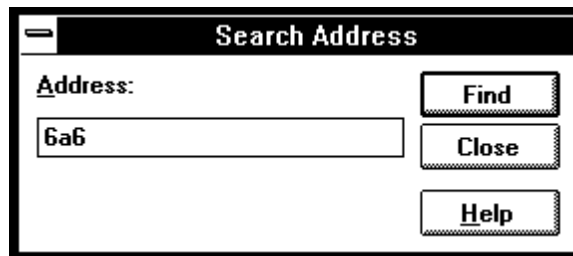
Address   Lets you enter the address to search for.

Find   Searches for the specified address.

Close   Closes the dialog box.

Command File Command

```
CUR(SOR) address
```
When used before the COME command, this command can be used to run to a particular address.

See Also

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

Search→Current PC (ALT, -, R, C)

Searches for, and displays, the location of the current program counter in the Source window.

Command File Command

`CUR(SOR) PC`
This command can be used to show the current PC in the Source window.

Search Directories Dialog Box

Choosing the Directories... button in the Select Source dialog box opens the
following dialog box:

Directory	Lets you enter the directory to be added to the source file search path.
Search Source Directories	Lists the directories in the source file search path.
Add	Adds the directory entered in the Directory text box to the source file search path.
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:
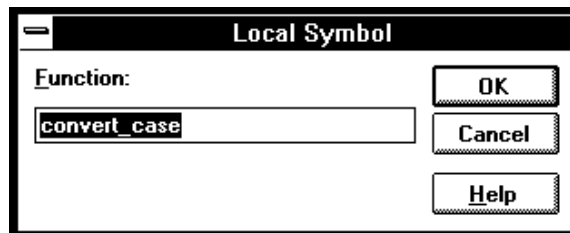
Function	Selects the function for which the local variable information is displayed.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

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:

```
┌─────────────────────────────────────────────────┐
│ ▬          Assembler Symbol                      │
│ Module:                              ┌─────────┐  │
│                                      │   OK    │  │
│ ┌────────────────────────────────┐   └─────────┘  │
│ │ sample                         │   ┌─────────┐  │
│ └────────────────────────────────┘   │ Cancel  │  │
│                                      └─────────┘  │
│                                      ┌─────────┐  │
│                                      │  Help   │  │
│                                      └─────────┘  │
└─────────────────────────────────────────────────┘
```
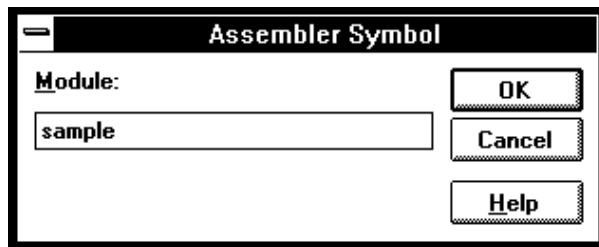
Module	Selects the module for which the local symbols are displayed.
OK	Executes the command and closes the dialog box.
Cancel	Cancels the command and closes the dialog box.

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:
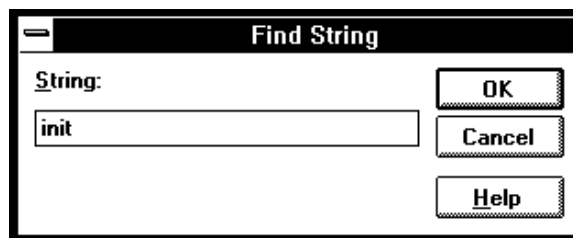
String   Specifies the string.

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:

```
┌─────────────────────────────────────────────────────┐
│ ▬         User Defined Symbol Add                     │
│ Symbol Name:                          ┌──────────┐    │
│                                       │    OK    │    │
│ imp_start                             └──────────┘    │
│                                       ┌──────────┐    │
│ Address:                              │  Cancel  │    │
│                                       └──────────┘    │
│ 494                                   ┌──────────┐    │
│                                       │   Help   │    │
│                                       └──────────┘    │
└─────────────────────────────────────────────────────┘
```
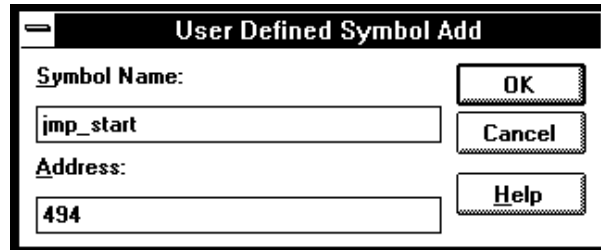
Symbol Name   Specifies the symbol to be added.

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
```
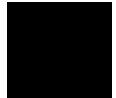
Trace Window Commands (Emulator Only)

This section describes the following commands:

- Display→Mixed Mode (ALT, -, D, M)

- Display→Source Only (ALT, -, D, S)

- Display→Bus Cycle Only (ALT, -, D, C)

- Display→Count→Absolute (ALT, -, D, C, A)

- Display→Count→Relative (ALT, -, D, C, R)

- Display→From State... (ALT, -, D, F)

- Display→Options→Dequeue ON (ALT, -, D, O, O)

- Display→Options→Dequeue OFF (ALT, -, D, O, 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→Mixed Mode (ALT, -, D, M)

Chooses the source/mnemonic mixed display mode.

Command File Command

```
TRA(CE) DIS(PLAY) MIX(ED)
```

See Also

"To display source doce mixed with assembly instructions" in the "Loading and Displaying Programs" section of the "Debugging Programs" chapter.

Display→Source Only (ALT, -, D, S)

Selects the source only display mode.

Command File Command

TRA(CE) DIS(PLAY) SOU(RCE)

See Also

"To display bus cycles" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Bus Cycle Only (ALT, -, D, C)

Selects the bus cycle only display mode.

Command File Command

TRA(CE) DIS(PLAY) BUS

See Also

"To display bus cycles" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.
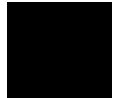
Display→Count→Absolute (ALT, -, D, C, A)

Selects the absolute mode (the total time elapsed since the trigger) for count information.

Command File Command

TRA(CE) DIS(PLAY) ABS(OLUTE)

See Also

"To display absolute or relative counts" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Count→Relative (ALT, -, D, C, R)

Selects the relative mode (the time interval between the current and previous cycle) for count information.

Command File Command

TRA(CE) DIS(PLAY) REL(ATIVE)

See Also

"To display absolute or relative counts" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→From State... (ALT, -, D, F)

Changes disassembly of bus cycle data in the Trace window.

Bus cycle data in the Trace window is always disassembled. However, because assumptions are made about where instructions start in the captured data, the disassembly may not always be correct. If you see disassembled information that does not look correct, you can use this command to change where the disassembly starts from.

Trace Disassemble From Dialog Box

Choosing the Display→From State... (ALT, -, D, F) command opens the following dialog box:

State   Enter the number of the state to start disassembly from. Select the Low Word option when you want to start disassembly from the low 16-bits of the 32-bits of captured data.

Dequeued Data   When trace data is being dequeued (in other words, when
Align From   captured states are shuffled so that operand cycles appear
State   with the instruction cycles that cause them), this box lets you enter the number of the operand cycle state that is caused by the instruction cycle state you are disassembling from.

OK   Disassembles and displays the trace data, and closes the dialog box.

Cancel   Cancels the command and closes the dialog box.

Command File Command

`MOD(E) TRA(CE) DIS(PLAY) FRO(M) state-num`
Specifies the state you want to disassemble from.

`MOD(E) TRA(CE) DIS(PLAY) HIG(WORD)/LOW(WORD)`
Specifies whether disassembly should start from the high or low 16-bits of the 32-bits of captured data.

`MOD(E) TRA(CE) DIS(PLAY) ALI(GN) state-num`
When trace data is being dequeued, this command specifies the first operand cycle state associated with the instruction cycle state you are disassembling from.

See Also

"To change the disassembly of bus cycle data" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Options→Dequeue ON (ALT, -, D, O, O)

Dequeues bus cycle data in the Trace window.

This command shuffles bus cycle states in the Trace window so that operand cycles immediately follow the instruction cycles that caused them. And, unexecuted instructions are removed from the display.

When dequeueing bus cycle data, ?TAKEN? may appear in disassembled branch instructions when the dequeuer is not able to determine whether the branch was taken. If, later in the trace list, you see the branch was taken, you may need to restart disassembly at the state that contains the branch destination (by using the Display→From State... (ALT, -, D, F) command in the Trace window's control menu).

Command File Command

MOD(E) TRA(CE) DIS(PLAY) DEQ(UEUE)

See Also

"To display dequeued trace data" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Display→Options→Dequeue OFF (ALT, -, D, O, F)

Turns OFF dequeueing of bus cycle data in the Trace window.

Command File Command

MOD(E) TRA(CE) DIS(PLAY) NOD(EQUEUE)

See Also

"To display dequeued trace data" in the "Tracing Program Execution" section of the "Debugging Programs" chapter.

Copy→Window (ALT, -, P, W)

Copies the information currently in the Trace window to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

TRA(CE) COP(Y) DIS(PLAY)

See Also

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

Copy→All (ALT, -, P, A)

Copies all the trace information to the specified listing file.

The listing file is specified with the File→Copy Destination... (ALT, F, P) command.

Command File Command

TRA(CE) COP(Y) ALL

Search→Trigger (ALT, -, R, T)

Positions the trigger state at the top of the Trace window.

Command File Command

TRA(CE) FIN(D) TRI(GGER)

Search→State... (ALT, -, R, S)

Positions the specified state at the top of the Trace window.

Search Trace State Dialog Box

Choosing the Search→State... (ALT, -, R, S) command opens the following
dialog box:
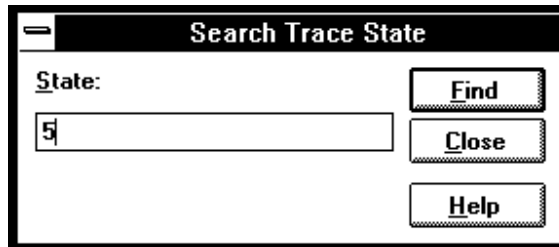
State   Lets you enter the trace state number to search for.

Find   Searches for the specified trace state.

Close   Closes the dialog box.

Command File Command

TRA(CE) FIN(D) STA(TE) state_num
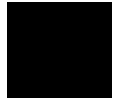
Trace Spec Copy→Specification (ALT, -, T, S)

Copies the current trace specification to the listing file.

Command File Command

TRA(CE) COP(Y) SPE(C)

Trace Spec Copy→Destination... (ALT, -, T, D)

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

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

Command File Command

COP(Y) TO filename

WatchPoint Window Commands

This section describes the following command:

- Edit...

Edit... (ALT, -, E)

Registers or deletes watchpoints.

Variables can be selected from the another window (in other words, copied to the clipboard) before choosing the Edit... (ALT, -, E) command from the WatchPoint window's control menu, and they will automatically appear in the dialog box that is opened.

Dynamic variables can be registered and displayed in the WatchPoint window when the current program counter is in the function in which the variable is declared. If the current program counter is not in the function, the variable name is invalid and results in an error.
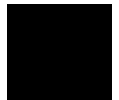
WatchPoint Dialog Box

Choosing the Edit... (ALT, -, E) command from the WatchPoint window's
control menu opens the following dialog box:
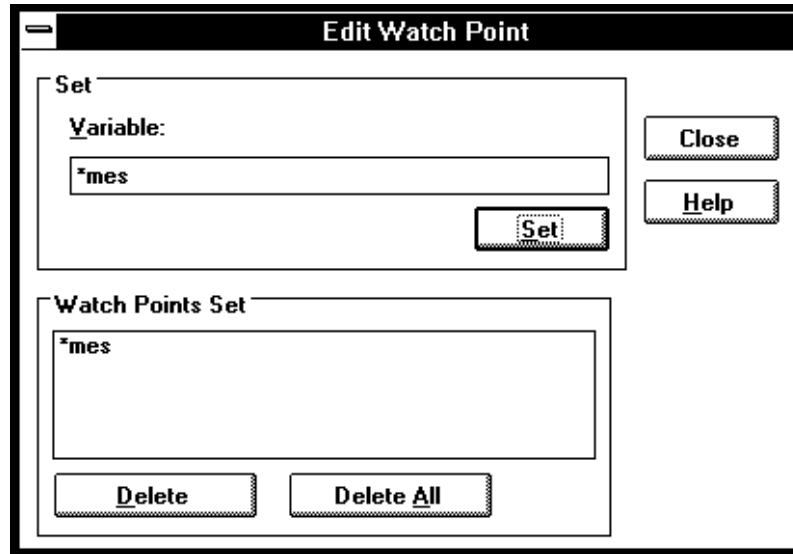
Variable	Lets you enter the name of the variable to be registered as a watchpoint. The contents of the clipboard, usually a variable selected from another window, automatically appears in this text box.
Watch Points Set	Lists the current watchpoints and allows you to select the watchpoint to be deleted.
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.

11

Window Pop-Up Commands

Window Pop-Up Commands

This chapter describes the commands that can be chosen from the pop-up menus in debugger windows. Pop-Up menus are accessed by clicking the right mouse button in the window.

- BackTrace Window Pop-Up Commands

- Source Window Pop-Up Commands

BackTrace Window Pop-Up Commands

- Source at Stack Level

Source at Stack Level

For the cursor-selected function in the BackTrace window, this command displays the function call in the Source window.

Source Window Pop-Up Commands

- Set Breakpoint

- Clear Breakpoint

- Evaluate It

- Add to Watch

- Run to Cursor

Set Breakpoint

Sets a breakpoint on the line containing the cursor. Refer to the
Breakpoint→Set at Cursor (ALT, B, S) command.

Clear Breakpoint

Deletes the breakpoint on the line containing the cursor. Refer to the
Breakpoint→Delete at Cursor (ALT, B, D) command.

Evaluate It

Evaluates the clipboard contents and places the result in the Expression
window. Refer to the Evaluate... (ALT, -, E) command available from the
Expression window's control menu.

Add to Watch

Adds the selected variable (that is, the variable copied to the clipboard) to the WatchPoint window. Refer to the Variable→Edit... (ALT, V, E) command.

Run to Cursor

Executes the program up to the Source window line containing the cursor. Refer to the Execution→Run to Cursor (ALT, R C) command.

12

Other Command File and Macro
Commands
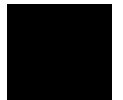
Other Command File and Macro Commands

This chapter describes the commands that are only available in command files, break macros, or buttons.

- BEEP

- EXIT

- FILE CHAINCMD

- FILE RERUN

- NOP

- TERMCOM

- WAIT

BEEP

Sounds beep during command file or break macro execution.

Command File Command

```
BEEP
```
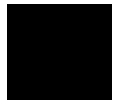
EXIT

Exits, or conditionally exits, command file execution.

Command File Command

```
EXIT
```
Exits command file execution.

```
EXIT VAR(IABLE) address value
```
Exits command file execution if the variable contains the value.

```
EXIT REG(ISTER) regname value
```
Exits command file execution if the register contains the value.

```
EXIT MEM(ORY) BYTE/WORD/LONG address value
```
Exits command file execution if the memory location contains the value.

```
EXIT IO BYTE/WORD address value
```
Exits command file execution if the I/O location contains the value.

FILE CHAINCMD

Chains command file execution.

This command lets you run one command file from another nonrecursively; in other words, control is not returned to the original command file.

By contrast, the FILE COMMAND command is recursive; if you use the FILE COMMAND command to run one command file from another, control will be returned to the original command file. FILE COMMAND commands can be nested four levels deep.

Command File Command

```
FILE CHAINCMD filename
```
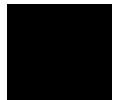
FILE RERUN

Starts command file execution over again.

This command is useful for looping stimulus files or running a demo or other command file continuously.

Command File Command

FILE RERUN

NOP

No operation.

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

Command File Command

```
NOP
```

```
NOP comments
```
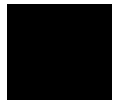
TERMCOM

Sends Terminal Interface commands to the HP 64700.

The HP 64700 Card Cage contains a low-level Terminal Interface, which allows you to control the emulator's functions directly. You can use the TERMCOM command to bypass the RTC Interface and send commands directly to the low-level Terminal Interface.

There is no window in the RTC Interface where you can execute TERMCOM commands directly. The only way to execute them with the RTC Interface is to make them part of a command file and then run the command file from an RTC Interface window.

You may need to start a unique target system that requires emulator intervention that is only available through the Terminal Interface. You can create the command file and then execute it at the appropriate time using a command such as File→Run Cmd File..., and place the name of your command file in the Run Command File dialog box.

The danger in using Terminal Interface commands via the TERMCOM command is that the RTC Interface may not be updated to know the state of the emulator. Some Terminal Interface commands can be executed by using the TERMCOM command, and the RTC Interface will not know that they were executed. Other Terminal Interface commands can be executed and the RTC Interface will be updated immediately. For example:

- If you have a command in your command file that changes the setting of RealTime→Monitor Intrusion→Disallowed/Allowed, (such as, TERMCOM "cf rrt=en"), the RTC Interface will not know about this change and will continue to try to operate according to the earlier setting. In this case, the RTC Interface may try to update its displays when the emulator is set to deny monitor access to the registers and memory.

- If you have a command in your command file that writes a value to memory (such as, TERMCOM "00000..00fff=0"), the Memory window will be updated immediately to show the new value, assuming you have chosen RealTime→Monitor Intrusion→Allowed.

Do not use the following Terminal Interface commands with the RTC
TERMCOM command:

- stty, po, xp: These commands will change the operation of the
  communications channel, and are likely to hang the RTC Interface.

- echo, mac: These commands may confuse the communications
  protocols in use in the channel.

- wait: The pod will enter a wait state, blocking access by the RTC
  Interface.

- init, pv: These will reset the emulator and end your session.

- t: This will confuse the functions of trace status polling and unload.

Refer to your "Terminal Interface User's Guide" for more information about
Terminal Interface commands.

Command File Command

```
TERMCOM "ti-command"
```
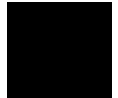
WAIT

Inserts wait delays during command file execution.

Command File Command

```
WAI(T) MON(ITOR)
```
Waits until MONITOR status.

```
WAI(T) RUN
```
Waits until RUN status.

```
WAI(T) UNK(NOWN)
```
Waits until UNKNOWN status.

```
WAI(T) SLO(W)
```
Waits until SLOW CLOCK status.

```
WAI(T) TGT(RESET)
```
Waits until TARGET RESET status.

```
WAI(T) SLE(EP)
```
Waits until SLEEP status.

```
WAI(T) GRA(NT)
```
Waits until BUS GRANT status

```
WAI(T) NOB(US)
```
Waits until NOBUS status.

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

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

```
WAI(T) TIM(E) seconds
```
Waits for a number of seconds.

13

Error Messages

Error Messages

This chapter helps you find details about the following error messages:

- Bad RS-232 port name
- Bad RS-422 card I/O address
- Could not open initialization file
- Could not write Memory
- Error occurred while processing Object file
- General RS-232 communications error
- General RS-422 communications error
- HP 64700 locked by another user
- HP 64700 not responding
- Incorrect DLL version
- Incorrect LAN Address (HP-ARPA, Windows for Workgroups)
- Incorrect LAN Address (Novell)
- Incorrect LAN Address (WINSOCK)
- Internal error in communications driver
- Internal error in Windows
- Interrupt execution (during run to caller)
- Interrupt execution (during step)
- Interrupt execution (during step over)
- Invalid transport name
- LAN buffer pool exhausted
- LAN communications error
- LAN MAXSENDSIZE is too small
- LAN Socket error
- Object file format ERROR
- Out of DOS Memory for LAN buffer
- Out of Windows timer resources
- PC is out of RAM memory
- Timed out during communications

Bad RS-232 port name

RS-232 port names must be of the form "COM<number>" where <number> is a decimal number from 1 to the number of communications ports within your PC.

Bad RS-422 card I/O address

The RS-422 card's I/O address must be a hexadecimal number from 100H through 3F8H whose last digit is 0 or 8 (example 100, 108, 110). Select an I/O address that does not conflict with the other cards in your PC.

Could not open initialization file

The initialization file was not found in the same directory where the executable file was found.

For example, if the application file is b3627.EXE, the initialization file b3627.INI is expected to be found in the same directory.

To fix this problem, you may be able to find the initialization file and move it to the same directory as the executable file, or you can create a new initialization file from the default initialization file. For example:

COPY b3627DEF.INI Bxxxx.INI

Note that the above command is the DOS COPY command. Do not use the ksh 'cp b3627DEF.INI Bxxxx.INI' command. Use only the DOS 'COPY b3627DEF.INI b3627.INI' command.

If you cannot find the default initialization file either, you can re-install the debugger software.

For correct operation, make certain the b3627.INI file has both read and write permission.

Could not write Memory

You may see this error message when trying to load a file or perform any other task that requires use of the monitor. The emulation monitor is used to load files, which requires writing to memory. If you have chosen RealTime→Monitor Intrusion→Disallowed the monitor will not be usable, and Execution→Reset may prevent use of the monitor in some emulators.

Choose RealTime→Monitor Intrusion→Allowed, and Execution→Break to ensure that the emulation monitor is running. The Status window should show Emulator: RUNNING IN MONITOR.

With this setup, the emulator should be able to write to Memory.

If you are still unable to load a file, select "Symbols Only" in the Load Object File dialog box and try to load the file. If Symbols Only will not load, the problem is in your symbols.

Choose "Data Only" in the Load Object File dialog box and try to load the file. If the symbols loaded, but the data fails to load, the problem is in your program code.

Call your local HP representative.

Error occurred while processing Object file

The following is a list of typical reasons why an error might occur while processing an object file. There are many other possible reasons.

- Bad record in the object file.

- File is in wrong format.

- File does not follow OMF Specifications correctly.

- No memory mapped.

- Attempt to write to guarded memory.

- Emulator restricted to real-time runs. Enter the command, "RealTime→Monitor Intrusion→Allowed".

- Emulator not executing the monitor. Enter the command, "Execution→Break".

Another message often occurs along with this message. View the help information for the other message, if available.

Call your local HP representative.
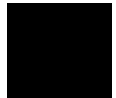
General RS-232 communications error

In general, these messages indicate that the RS-232 communication has intermittent errors. Sometimes you will get this message if you power on the emulator, or when you try to connect to the emulator. In that case, simply retry the connection (by double-clicking on the RS232C driver line in the selection box); if you connect with no problems the second time, you can ignore the original message.

If you get this message other than during connection, you can try to fix the problem by:

- Reducing the length of the RS-232 cable between the PC and the HP 64700.

- Reducing the number of tasks running under Windows.

- Reducing the baud rate (the default is 19200).

For further information, refer to the paragraph titled, "If you have RS-232 connection problems" in the Communications Help screen, or in Chapter 15, "Installing the Debugger" in the Real-Time C Debugger User's Guide.

General RS-422 communications error

In general, these messages indicate that the RS-422 communication has intermittent errors. Sometimes you will get this message if you power on the emulator, or when you try to connect to the emulator. In that case, simply retry the connection (by double-clicking on the HP-RS422 driver line in the selection box); if you connect with no problems the second time, you can ignore the original message.

If you get this message other than during connection, you can try to fix the problem by:

- Reducing the number of tasks running under Windows.

- Reducing the baud rate (the default is 230400).

HP 64700 locked by another user

Because it is possible to destroy another user's measurement by choosing the Unlock button in the error dialog box, check with the other user before unlocking the HP 64700.

Note that if the other user is actually using an interface to the HP 64700, an Unlock request will fail.

HP 64700 not responding

The HP 64700 has not responded within the timeout period. There are several possible causes of this error. For example, a character could have dropped during RS-232 communications, or some network problem could have disrupted communications.

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

See also: The description for the error message titled, "Timed out during communications."

Incorrect DLL version

The version of the dynamic link libraries (.DLLs) used by the Real-Time C Debugger does not match the version of the main program (.EXE).

If you have two versions of the debugger on your system, you may see this message when you try to execute both of them at the same time, or when you execute one version and then the other without restarting Windows. Once DLLs have been loaded into Windows memory, they stay there until you exit Windows. Therefore, exit windows, restart windows, and try again.

This message will also appear if you have somehow loaded a version of the DLLs that is different from the version of the executable. In this case, you must reload your software.

Incorrect LAN Address (HP-ARPA, Windows for Workgroups)

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

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

```
15.6.28.0
```

A hostname is a name that is related (mapped) to an IP address by a database. For example, the file \LANMAN.DOS\ETC\HOSTS (HP-ARPA) or \WINDOWS\HOSTS (Windows for Workgroups) may contain entries of the form:

```
system1 15.6.28.0
```

Note

The directory of the "hosts" file may be different on your system.

If "HP Probe" or "DNR" (Domain Name Resolution) is available on your PC, those are consulted first for a mapping between the hostname and the IP address. If the hostname is not found by that method, or if those services are unavailable, the local "hosts" file is consulted for the mapping.

Note that if "Probe" is available on your system but unable to resolve the address, there will be a delay of about 15 seconds while Probe is attempting to find the name on the network.

Incorrect LAN Address (Novell)

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

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

`15.6.28.0`

A hostname is a name that is related (mapped) to an IP address by a database. For example, the file \NET\TCP\HOSTS may contain entries of the form:

`system1 15.6.28.0`

Note

The directory of the "hosts" file may be different on your system. Also, all files defined by the PATH TCP_CFG setting under "Protocol TCPIP" in the NET.CFG files are searched.

Incorrect LAN Address (WINSOCK)

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

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

`15.6.28.0`

A hostname is a name that is related (mapped) to an IP address by a database. For example, the hosts file may contain entries of the form:

`system1 15.6.28.0`

Note

Because WINSOCK is a standard interface to many LAN software vendors, you need to read your LAN vendor's documentation before specifying the LAN address.
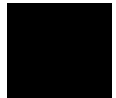
Internal error in communications driver

These types of errors typically occur because other applications have used up a limited amount of some kind of global resource (such as memory or sockets).

You usually have to reboot the PC to free the global resources used by the communications driver.

Internal error in Windows

These types of errors typically occur because other applications have used up a limited supply of some kind of global resource (such as memory, sockets, tasks, or handles).

You usually have to reboot the PC to free the global resources used by Windows.

Interrupt execution (during run to caller)

The Return dialog box appears when running to the caller of a function and the caller is not found within the number of milliseconds specified by StepTimerLen in the .INI file of the debugger application.

You can cancel the run to caller command by choosing the STOP button, which causes program execution to stop, the breakpoint to be deleted, and the processor to transfer to the RUNNING IN USER PROGRAM status.

Interrupt execution (during step)

The Step dialog box appears when stepping a source line or assembly instruction and the source line or instruction does not execute within the number of milliseconds specified by StepTimerLen in the .INI file of the debugger application.

You can cancel the step command by choosing the STOP button, which causes program execution to stop, the breakpoint to be deleted, and the processor to transfer to the RUNNING IN USER PROGRAM status.

Interrupt execution (during step over)

The Step dialog box appears when stepping over a function or subroutine and the function or subroutine does not execute within the number of milliseconds specified by StepTimerLen in the .INI file of the debugger application.

You can cancel the step-over command by choosing the STOP button, which causes program execution to stop, the breakpoint to be deleted, and the processor to transfer to the RUNNING IN USER PROGRAM status.

Invalid transport name

The transport name chosen does not match any of the possible transport names (RS232C, HP-ARPA, Novell-WP, WINSOCK1.1, W4WG-TCP, or HP-RS422).

The transport name can be specified either on the command line with the -t option or in the .INI file:

[Port]
Transport=<transport name>

Choosing an appropriate transport in the dialog box that follows this error message will correct the entry in the .INI file, but if the error is in the command line option, you must modify the command line (by using the "Properties..." command in the Program Manager).

LAN buffer pool exhausted

The LAN buffer pool is used as a temporary buffer between the time the debugger sends data and the time the LAN actually sends it. When this pool is exhausted, the debugger cannot send any data across the LAN.

The size of the sockets buffer pool is configured in the network installation procedure. The size and number of LAN buffer pools can be changed by editing your network configuration file.

LAN communications error

This message may appear after any kind of LAN error.

Refer to the documentation for your LAN software for descriptions of the types of problems that can cause LAN errors.

LAN MAXSENDSIZE is too small

This message indicates you have configured your LAN with a value or MAXSENDSIZE that is less than 100 bytes. Note that the default is 1024 bytes.

The Real-Time C Debugger requires at least 100 bytes for this parameter.

To fix this, change the following entry in your PROTOCOL.INI file and reboot your PC:

[SOCKETS]
MAXSENDSIZE

LAN socket error

A TCP-level error has occurred on the network. See your network administrator.
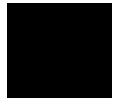
Object file format ERROR

This message is typically caused by one of two conditions:

- Bad format file. Perhaps there is a bad record within the file. If you have a file format verifier, submit your file to it to determine whether or not all records are in the correct format.

- Unknown construct. Perhaps the construct of your file is unfamiliar to the reader.

To respond to this error message, verify the file format, and ensure that the reader can understand the file format in use.

If these steps do not solve the problem, call your local HP representative.

Out of DOS Memory for LAN buffer

This means that there is not enough memory in the lower 1 Mbyte of address space (that is, conventional memory) for the LAN driver to allocate a buffer to communicate with the LAN TSR.

When you are in windows, and execute the DOS command "mem", you cannot see the memory that is in the lower 1 Mbyte that is used by the windows program. If you have the Microsoft program "heapwalker", you can use it to see what programs have allocated space in the address range 0 through FFFFF.

To fix this, you can:

- Reduce the number of TSRs running on your PC (before Windows starts) that use conventional memory.

- Reconfigure your network to have fewer sockets or modules loaded, or to be configured for fewer total connections.

- Use a different memory manager to reduce your network memory usage, such as QEMM.

Out of Windows timer resources

The debugger is not able to acquire the timer resources it needs.

There are a limited number of timer resources in Windows. You may be able to free timer resources by closing other applications.

PC is out of RAM memory

The debugger is not able to acquire the memory it needs because other applications are using it, or because of fragmented memory.

You may be able to free memory by closing other applications, or you might have to reboot the PC to cause memory to be unfragmented.

Timed out during communications

The HP 64700 has not responded within the timeout period. There are
various causes for this error. For example, a character could have been
dropped during RS-232 communications or some network problem could
have disrupted communications.

The timeout period for reading and writing to the HP 64700 is defined by
TimeoutSeconds in either the [RS232C], [HP-ARPA], [Novell-WP], or
[HP-RS422] section of the b3627.INI file. For example, if you are using the
RS-232C transport:

```
[RS232C]
TimeoutSeconds=<seconds>
```

The number of seconds can be between 1 and 32767. The default is 20
seconds.

If you are using RS-232C or RS-422 transport ...

The TimeoutSeconds value is also used for connecting to the HP 64700 (as
well as for reading and writing).

If you are using HP-ARPA or Novell-WP transport ...

If there are several gateways or bridges between the PC and the emulator,
larger values of TimeoutSeconds may be reasonable.

The timeout period for connecting to the HP 64700 is defined in the
PROTOCOL.INI file.

```
[TCPIP_XFR]
TCPCONNTIMEOUT=<seconds>
```

The default connection timeout is 30 seconds.

The remainder of this discussion shows you how to overcome the problem of
"connection timed out" during large memory fill operations.

The RTC interface sends the memory fill operation to the emulator as a single
command. While the command is executing in the emulator, the emulator
cannot respond to inquiries from the interface about its status. If the
memory fill takes long enough, the connection will time out.

Emulators for some microprocessors take up to one minute per megabyte to perform a memory fill operation. Timeout default values for RTC interfaces shipped from HP are typically 45 seconds.

First Workaround. Modify the TimeoutSeconds field (discussed above) to increase the TimeoutSeconds value. Then exit the interface and restart it (to ensure that the new value of TimeoutSeconds is read). You may experiment with several values of TimeoutSeconds to find the value that allows you to do a memory fill. The problem with this workaround is that all timeouts will take this new longer time, and you may find this annoying when you are not doing memory fill operations.

Second Workaround. Create a command file that contains TERMCOM commands to write to small portions of the overall memory to be filled. For example, suppose the following Memory window command causes the emulator to time out, "Memory→Utilities→Fill→0 to ffff".

You might make a command file named memfill.cmd, and place the following commands in it:

```
TERMCOM "m 00000..00fff=0"
TERMCOM "m 01000..01fff=0"
TERMCOM "m 02000..02fff=0"
TERMCOM "m 03000..03fff=0"
TERMCOM "m 04000..04fff=0"
TERMCOM "m 05000..05fff=0"
TERMCOM "m 06000..06fff=0"
TERMCOM "m 07000..07fff=0"
TERMCOM "m 08000..08fff=0"
TERMCOM "m 09000..09fff=0"
TERMCOM "m 0a000..0afff=0"
TERMCOM "m 0b000..0bfff=0"
TERMCOM "m 0c000..0cfff=0"
TERMCOM "m 0d000..0dfff=0"
TERMCOM "m 0e000..0efff=0"
TERMCOM "m 0f000..0ffff=0"
```

When you choose File→Run Cmd File→... and select your memfill.cmd file, it will not exceed the timeout value. This is because the emulator will be able to respond to inquiries from the interface between execution of each of the TERMCOM commands in your command file.

Part 4

Concept Guide

Topics that explain concepts and apply them to advanced tasks.

Part 4
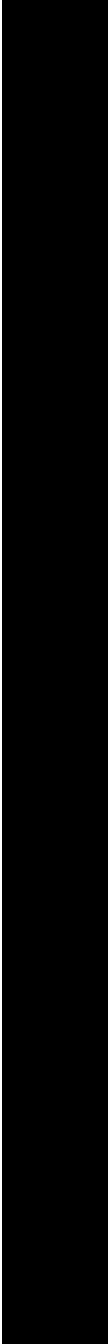
14

Concepts
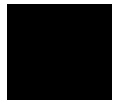
Concepts

This chapter describes the following topics.

- Debugger Windows
- Compiler/Assembler Specifications
- Trace Signals and Predefined Status Values

Debugger Windows

This section describes the following debugger windows:

- BackTrace

- Button

- Expression

- I/O

- Memory

- Register

- Source

- Status

- Symbol

- Trace

- WatchPoint
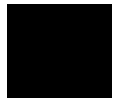
The BackTrace Window

The BackTrace window displays the function associated with the current program counter value and this function's caller functions in backward order. Applicable addresses are prefixed with module\#linenum information. The current arguments of these functions are also displayed.
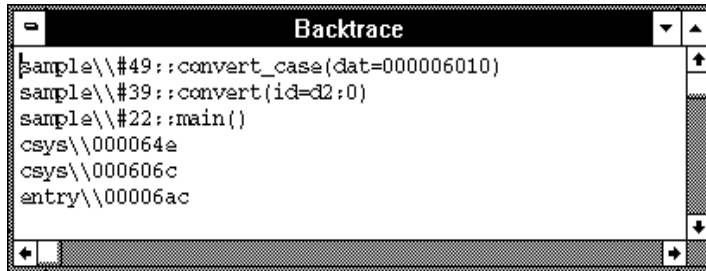
The BackTrace window is updated when program execution stops at an occurrence of breakpoint, break, or Step command.

The BackTrace window lets you copy text strings, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

By clicking the right mouse button in the BackTrace window, you can access the Source at Stack Level 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.
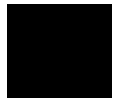
The Expression Window

The Expression window displays the results of the EVALUATE commands in command files or break macros.
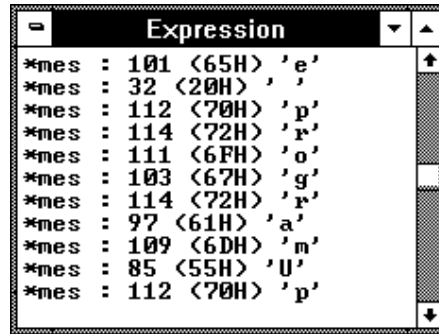
When a variable name is specified with the EVALUATE command, the Expression window displays the evaluation of the variable. When a quoted string of ASCII characters is specified with the EVALUATE command, the Expression window displays the string.

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.
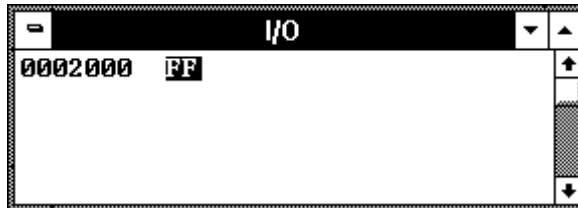
You can modify the contents of I/O locations by double-clicking on the value, using the keyboard to type in the new value, and pressing the Enter key.

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.

See Also

"Displaying and Editing I/O Locations" in the "Debugging Programs" chapter.

"I/O Window Commands" in the "Window Control Menu Commands" chapter.
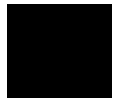
The Memory Window

The Memory window displays memory contents.

```
┌──────────────────────────────────────────────┬───┬───┐
│ ▬            Memory                         ▼ │ ▲ │
├──────────────────────────────────────────────┼───┤
│0000060100x 00 00 00 00 54 68 69 73  ....This │ ↑ │
│0000060180x 20 69 73 20 61 20 73 61   is a sa │   │
│0000060200x 6D 70 6C 65 20 70 72 6F  mple pro │   │
│0000060280x 67 72 61 6D 00 00 00 00  gram.... │   │
│0000060300x 00 00 00 00 00 00 00 00  ........ │   │
│0000060380x 00 00 00 00 00 00 00 01  ........ │   │
│0000060400x 75 50 50 45 52 20 41 4E  uPPER AN │   │
│0000060480x 44 20 6C 4F 57 45 52 20  D lOWER  │   │
│0000060500x 63 41 53 45 20 63 4F 4E  cASE cON │ ▯ │
│0000060580x 56 45 52 53 49 4F 4E 00  VERSION. │   │
│0000060600x 00 00 00 00 00 00 00 00  ........ │   │
│0000060680x 00 00 00 00 00 00 00 00  ........ │   │
│0000060700x 00 00 62 A4 00 00 00 00  ..b..... │   │
│0000060780x 00 00 00 00 00 00 00 00  ........ │   │
│0000060800x 00 00 00 00 00 00 00 00  ........ │   │
│0000060880x 00 00 00 00 00 00 00 00  ........ │   │
│0000060900x 00 00 00 00 00 00 00 00  ........ │ ↓ │
└──────────────────────────────────────────────┴───┘
```

The Memory window has *control menu* commands that let you change the
format of the memory display and the size of the locations displayed or
modified. When the absolute (single-column) format is chosen, symbols
corresponding to addresses are displayed. When data is displayed in byte
format, ASCII characters for the byte values are also displayed.

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

The Memory window contents are updated periodically when the processor is
running the user program.
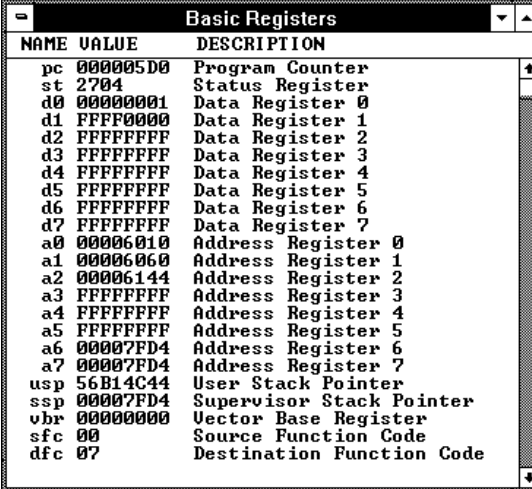
If a location is in target system memory, a temporary break from the user
program into the monitor program must occur in order for the debugger to
update or modify that location's contents. If it's important that the user
program execute without these types of interruptions, you should disallow
monitor intrusion. Even when monitor intrusion is allowed, you can stop
temporary breaks during the window update by turning polling OFF.

See Also

"Displaying and Editing Memory" in the "Debugging Programs" chapter.

"Memory Window Commands" in the "Window Control Menu Commands" chapter.

The Register 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.

```
┌─────────────────────────────────────────────┐
│ ▬          Basic Registers            ▼  ▲  │
│ NAME VALUE        DESCRIPTION                 │
│   pc 000005D0  Program Counter              ▲│
│   st 2704      Status Register               │
│   d0 00000001  Data Register 0               │
│   d1 FFFF0000  Data Register 1               │
│   d2 FFFFFFFF  Data Register 2               │
│   d3 FFFFFFFF  Data Register 3               │
│   d4 FFFFFFFF  Data Register 4               │
│   d5 FFFFFFFF  Data Register 5               │
│   d6 FFFFFFFF  Data Register 6               │
│   d7 FFFFFFFF  Data Register 7               │
│   a0 00006010  Address Register 0            │
│   a1 00006060  Address Register 1            │
│   a2 00006144  Address Register 2            │
│   a3 FFFFFFFF  Address Register 3            │
│   a4 FFFFFFFF  Address Register 4            │
│   a5 FFFFFFFF  Address Register 5            │
│   a6 00007FD4  Address Register 6            │
│   a7 00007FD4  Address Register 7            │
│  usp 56B14C44  User Stack Pointer            │
│  ssp 00007FD4  Supervisor Stack Pointer      │
│  vbr 00000000  Vector Base Register          │
│  sfc 00        Source Function Code          │
│  dfc 07        Destination Function Code    ▼│
└─────────────────────────────────────────────┘
```

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 windows' contents are updated periodically when the processor is running the user program and monitor intrusion is allowed.

A temporary break from the user program into the monitor program must occur in order for the debugger to update or modify register contents. If it's important that the user program execute without these types of interruptions, you should disallow monitor intrusion.

See Also

"Displaying and Editing Registers" in the "Debugging Programs" chapter.

"Register Window Commands" in the "Window Control Menu Commands" chapter.

The Source Window

The Source window displays source files, optionally with disassembled instructions intermixed.

The Source window contains a cursor whose position is used when setting or deleting breakpoints or break macros or when running the program up to a certain line.

The Source window lets you copy strings, usually variable or function names to be used in commands, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

The Source window also provides commands in the *control menu* that let you select whether disassembled instruction mnemonics should appear intermixed with the C source code.

By clicking the right mouse button in the Source window, you can also access pop-up menu commands.

```
┌─────────────────────────────────────────────────────────────────────┐
│ ▭                               Source                          ▼ ▲ │
├─────────────────────────────────────────────────────────────────────┤
│Filename : c:\hp\rtc\m360\demo\sample\sample.c                       │
├─────────────────────────────────────────────────────────────────────┤
││    #0018        {                                                 ↑│
│     0000004aa@x 0x4e560000 LINK.W    A6,#$0000                      │
│     #0019          init_data();                                    │
│     0000004ae@x 0x4eba002a JSR       ($04DA,PC)                     │
│     0000004b2@x 0x4e71     NOP                                      │
│     #0020          while(1)                                         │
│     #0021          {                                                │
│  BP #0022              convert(message_id);                         │
│  BP 0000004b4@x 0x2f390000 MOVE.L    $0000600C,-(A7)                │
│     0000004ba@x 0x4eba0060 JSR       ($051C,PC)                     │
│     0000004be@x 0x4e71     NOP                                      │
│     #0023.1    EVALUATE message_id                                  │
│     #0023.2    RUN                                                  │
│  BP #0023              message_id = next_message(message_id);       │
│     0000004c0@x.1    EVALUATE message_id                            │
│     0000004c0@x.2    RUN                                            │
├─────────────────────────────────────────────────────────────────────┤
│ ← ▭                                                              → ↓│
└─────────────────────────────────────────────────────────────────────┘
```

Filename	The name of the displayed source file appears at the top of the window.
Source Lines	C source code is displayed when available. Source lines are preceded by the corresponding line numbers.
	When programs are written in assembly language or when no C source code is available, disassembled instruction mnemonics are displayed.
	The interface will only support display in either trace or source windows of source lines numbered less than 32,000.
Disassembled Instructions	In the Mnemonic Display mode, disassembled instruction mnemonics are intermixed with the source lines. Disassembled lines contain address, data, and mnemonic information.
	When symbolic information is available for the address, the corresponding symbol line precedes the disassembled instruction, displayed in the module_name\\symbol_name format.
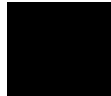
Current PC	The line associated with the current program counter is highlighted.
Scroll Bars	For C source files, the display scrolls within the source files. For assembly language programs or programs for which no source code is available, the display scrolls for all the memory space.
"BP" Marker	The breakpoint marker, BP, appears at the beginning of the breakpoint lines or break macro lines.
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 Pop-Up Commands" in the "Window Pop-Up Commands" chapter.

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

The Status Window

The Status window shows:

- Emulator status.

- Trace status.

- Scope of the current program counter value.

- Progress of symbols being loaded from a file.

- Last five asynchronous messages from the emulator.

```
┌─────────────────────────────────────────────┐
│ ▭              Status              ▼ ▲ │
├─────────────────────────────────────────────┤
│ EMULATOR: RUNNING IN MONITOR                 │
│ TRACE:    HALTED                             │
│ SCOPE:    NONE                               │
│ SYMBOL:   FILE LOAD COMPLETE                 │
│ MESSAGES:                                    │
│                                              │
│                                              │
└─────────────────────────────────────────────┘
```
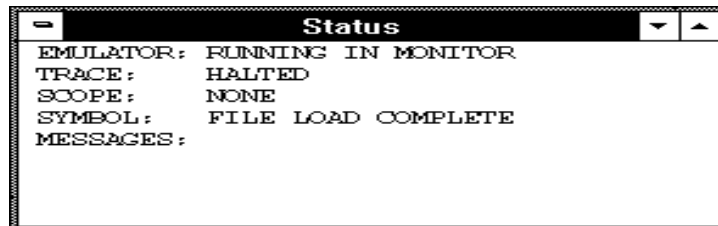
Emulation Processor Status Messages

BUS GRANT TO TARGET SYSTEM DEVICE
The bus is granted to some device in the target system.

EMULATION RESET
The emulation processor is being held in the reset state by the emulator.

EMULATION RESET BY TARGET
The emulation processor is being held in a reset state by a RESET signal from the target system.

DOUBLE BUS FAULT
A double bus fault ocurred because of multiple address or bus errors. When this happens, the emulator breaks into the monitor.

HALTED
The emulation processor has halted.
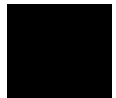
```
HUNG BUS CYCLE
```
A hung bus cycle occurred during a memory access operation. This message indicates the emulator detected the hung bus cycle and terminated it. When this happens, retry the command that caused the hung bus cycle. You may need to determine the source of the termination (such as the processor, emulation memory, or target memory) and make the corrections required.

```
IN DMA
```
The bus is granted to a device performing DMA.

```
NO BUS CYCLES
```
This status may indicate that the emulation processor is running code in its internal memory for long periods of time. Otherwise, the bus cycle is too slow or no bus cycle is provided.

```
NO DSACK OR BERR
```
The emulator has stopped in the middle of a bus cycle and is in a wait state. Look at the address and type of cycle and make sure the target system provides a bus cycle termination at this address. If it does, the most likely cause is either that the target system missed the "start of cycle" indication from the emulator or that the emulator missed the "cycle termination" indication from the target system.

```
NO TARGET POWER
```
If this status remains after target system powerup, check the mechanical installation of the probe, check the target system power supply voltage, or check for blown fuses.

```
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.

SLOW CLOCK
No proper clock pulse is supplied from the external clock.

UNKNOWN STATE
The emulation processor is in an unknown state.

WAITING FOR CMB
The emulator has been set up for coordinated starting and stopping with
other emulators. It is waiting for one of the emulators to drive the
Coordinated Measurement Bus (CMB) EXECUTE line.

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.

Other Emulator Status Messages

The Status window may also contain status messages other than the
emulation processor status messages described above:

ACCESS TO GUARD BREAK
Program execution has stopped due to a write to a location mapped as
guarded memory.

BREAKPOINT HIT AT address
The breakpoint specified in the assembled line was hit and program
execution stopped at "address".

BREAKPOINT 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".

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).

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 a location mapped as ROM.
These types of breaks must be enabled in the emulator configuration.
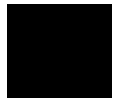
Trace Status Messages

COMPLETE
The trace completed because the trace buffer is full. The results are
displayed in the Trace window.

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.

RUNNING | CAPTURING
The trace has been started and the trigger condition has occurred, but 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.

RUNNING | WAITING FOR TRIGGER
The trace has been started, but the trigger condition has not occurred.

The Symbol Window

The Symbol window displays information on the following types of symbols:

- Modules
- Functions
- Global symbols
- Local symbols
- Global Assembler symbols
- Local Assembler symbols
- User-defined symbols

The Symbol window has *control menu* commands that let you display various types of symbols, add or delete user-defined symbols, copy Symbol window information, or search for symbols that contain a particular string.

The Symbol window lets you copy symbols to the clipboard by clicking the left mouse button. The symbol information can then be pasted from the clipboard in other commands.

```
┌─────────────────────────── Symbol ──────────────────▼─▲─┐
│ Functions                                                │
├──────────────────────────────────────────────────────┬──┤
│ change_status      int            0000005b8..0000005cb │▲ │
│ convert            int            00000051c..00000057f │  │
│ convert_case       int            000000580..0000005b7 │▓ │
│ init_data          int            0000004da..00000051b │▓ │
│ main               int            0000004aa..0000004d9 │  │
│ next_message       int            0000005cc..0000005df │  │
│                                                        │  │
│                                                        │▼ │
└──────────────────────────────────────────────────────┴──┘
```
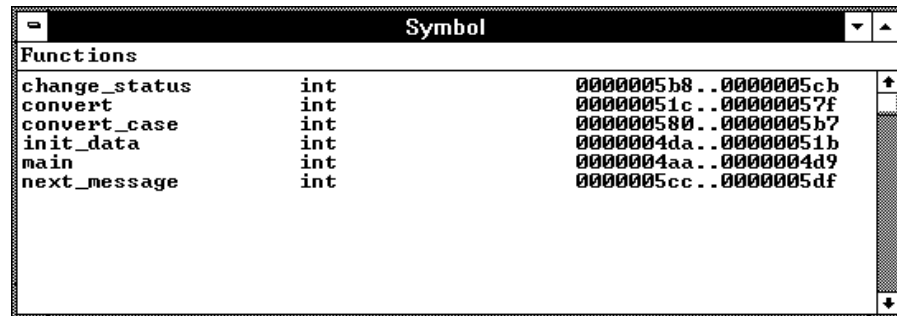
Symbols are displayed with "type" and "address" values where appropriate.

See Also

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

"Symbol Window Commands" in the "Window Control Menu Commands" chapter.

The Trace Window (Emulator Only)

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 show absolute or relative, or copy information from the window.

The Trace window opens automatically when a trace is complete.

```
▭                    Trace - Loading Data [ Frame: 1]                    ▼  ▲
state typ module\\#line   :function       source                      <<1   >>
       0 SEQ sample\\#0054  :change_statu {                    ---------- TG  ↑
       2 SEQ sample\\#0062  :next_message {                    6.680 uS^TG
       4 SEQ sample\\#0038  :convert      {                    4.400 uS
       6 SEQ sample\\#0044  :convert_case {                    3.800 uS
       8 SEQ sample\\#0054  :change_statu {                    90.60 uS
      10 SEQ sample\\#0062  :next_message {                    6.680 uS
      12 SEQ sample\\#0038  :convert      {                    4.200 uS
      14 SEQ sample\\#0044  :convert_case {                    3.800 uS
      16 SEQ sample\\#0054  :change_statu {                    63.56 uS
      18 SEQ sample\\#0062  :next_message {                    6.880 uS
      20 SEQ sample\\#0038  :convert      {                    4.400 uS
      22 SEQ sample\\#0044  :convert_case {                    3.800 uS
      24 SEQ sample\\#0054  :change_statu {                    82.20 uS
      26 SEQ sample\\#0062  :next_message {                    6.880 uS
←                                                                      → ↓
```

For each line in the Trace window, the trace buffer state number, the type of state, the module name and source file line number, the function name, the source line, and the time count information are displayed.

The << and >> buttons let you move between the multiple frames of trace data that are available with newer analyzers for the HP 64700.

The type of state can be a sequence level branch (SEQ), a state that satisfies the prestore condition (PRE), or a normal state that matches the store conditions (in which case the type field is empty).

Bus cycle states show the address and data values that have been captured as well as the disassembled instruction or status mnemonics.

On startup, the system defaults to the source only display mode, where only source code lines are displayed. The source/bus cycle mixed display mode can be selected by using the Trace window control menu's Display→Mixed

Mode (ALT, -, D, M) command. In the source/bus cycle mixed display mode, each source code line is immediately followed by the corresponding bus cycles.

The trace buffer stores bus cycles only. The system displays source lines in the Trace window based on execution bus cycles.

See Also

"Tracing Program Execution" and
"Setting Up Custom Trace Specifications" in the "Debugging Programs" chapter.

"Trace Window Commands" in the "Window Control Menu Commands" chapter.
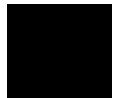
The WatchPoint Window

The WatchPoint window displays the contents of variables that have been registered with the Variable→Edit... (ALT, V, E) command or with the Edit... (ALT, -, E) command in the WatchPoint window's control menu.
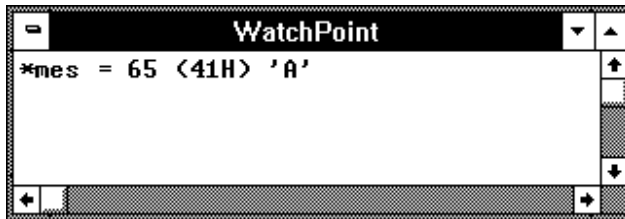
The contents of dynamic variables are displayed only when the current program counter is in the function in which the variable is declared.

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

The WatchPoint window lets you copy text strings, to the clipboard by double-clicking words or by holding down the left mouse button and dragging the mouse pointer.

See Also

"Displaying and Editing Variables" in the "Debugging Programs" chapter.

"WatchPoint Window Commands" in the "Window Control Menu Commands" chapter.

Compiler/Assembler Specifications

This section describes:

- IEEE-695 Object Files

- Compiling Programs with MCC68K

- Compiling Programs with AxLS

IEEE-695 Object Files

This section addresses the IEEE-695 object files compiled or assembled with the following compilers and assemblers:

- Microtec MCC68K Compiler

- Microtec ASM68K Assembler

- HP AxLS Compiler

- HP AxLS Assembler

Assembly Language Source File Display

The IEEE-695 object files do not contain assembly language source file information. Instead, memory contents are disassembled.

Mnemonic Display

An assembly language instruction preceding or following a function entry point may have multiple corresponding source code lines. For this type of instruction, the Source window in the Mnemonic Display mode shows multiple corresponding disassembled lines having the same address.

Single-Stepping Loop Control Statements

The system may fail in single-stepping such loop control statements as "while", "for", or "do while" statement.
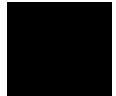
Pragma Statement and Debugger Display

When a "pragma" statement is used to describe an assembly language instruction in C source files, the source information is generated as follows in the IEEE-695 object files:

- A pragma instruction has a single line number.

- The address for the pragma instruction indicates the address for the first line of the instruction.

- The line number for the pragma instruction indicates the line number for the last line of the instruction.

This imposes the following display restriction on the Real-Time C Debugger:

The Source window in the Mnemonic Display mode shows lines in a pragma instruction all at one time as listed below.

```
#0010       #pragma asm
#0011           nop
#0012           nop
#0013       #pragma endasm
0001000     00          NOP
0001001     00          NOP
```

During single-stepping, the last line of the pragma instruction is highlighted while the program counter indicates the first line.

```
#0010       #pragma asm
#0011           nop
#0012           nop
#0013       #pragma endasm
```

Program counter indicating line 11

Highlighted line 12

Only the last line of the pragma instruction is displayed in the trace results.

Compiling Programs with MCC68K

1 Compile the source files with the mcc68k command.

2 Assemble the source files with the asm68k command.

3 Link the object files with the lnk68k command.

Required Compiler/Assembler/Linker

Compiler   Microtec MCC68K Compiler

Assembler   Microtec ASM68K Assembler

Linker   Microtec LNK68K Linker

Compiling

For compiling, use the mcc68k command in your Microtec C Compiler with the following option switches:

-g	Outputs debugging information.
-Gf	Generates fully-qualified path names for input files.
-nOg	Disables global flow optimization.
-nOR	Disables register variables.
-Kf	Creates frame pointers for functions.

Note   The -nOg and -nOR options allow the debugger to display arguments during backtracing.

Note   The -Kf option allows the debugger to trace function flow.
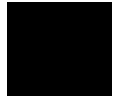
Assembling

For assembling, use the asm68k command in your Microtec Assembler with the following option switch:

-fd   Creates local symbols.

Linking

For linking, use the lnk68k command in your Microtec Linker. Specify the IEEE-695 file format for the load module.

Example

To compile and link sample.c user program into a load module, execute the following command, where sample.k is the linker command file:

```
A> mcc68k -g -Gf -Kf -nOg -nOR -l -esample.k -osample.x
sample.c -Wl,-m > sample.lst
```

Compiling Programs with AxLS

1 Compile the source files with the cc68k command.

2 Assemble the source files with the as68k command.

3 Link the object files with the ld68k command.

Required Compiler/Assembler/Linker

Compiler   HP AxLS CC68K Compiler

Assembler   HP AxLS AS68K Assembler

Linker   HP AxLS LD68K Linker

Compiling

For compiling, use the cc68k command in your HP AxLS C Compiler with the following option switches:

-Wc,-F   Disables register variables.

Note   The -Wc,-F option allows the debugger to display arguments during backtracing.

Assembling

For assembling, use the as68k command in your HP AxLS Assembler without any option switch.

Linking

For linking, use the ld68k command in your HP AxLS Linker. Specify the IEEE-695 file format for the load module.

Note   The Real-Time C Debugger does not support simulated I/O locations. You can use the -N compiler option to use a linker command file that does not include the simulated I/O library.

Example   To compile and link sample.c user program into a load module, execute the following command, where sample.k is the linker command file:

```
cc68k -p CPU32 -N -Wc,-F -Lix -k sample.k -o sample.x
sample.c
```
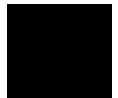
Trace Signals and Predefined Status Values

This section describes how emulation analyzer trace signals are assigned to
microprocessor address bus, data bus, and control signals.

Emulation Analyzer Trace Signals

```
Trace
Signals    Signal Name          Signal Description
-------    -----------    -------------------------------------
0-31       A0-A31         Address Lines 0-31

32-63      D0-D31         Processor Data 0-31

64-79                     Control Signals:
           Status 0       Reserved.
           Status 1..3    CPU function code FC0..FC2, respectively.
           Status 4       R/W from CPU.
           Status 5..6    SIZ0..SIZ1, respectively.
           Status 7..8    DSACK0..DSACK1, respectively.
           Status 9       Bus error (active low).
           Status 10      CPU halt line (active low).
           Status 11      Show cycle (if enabled, active low).
           Status 12      Flush (1st fetch following pipeline flush, active low).
           Status 13      External DMA cycle when low.
           Status 14      CPU function code FC3.
           Status 15      Fetch (active low).
```

Predefined Status Values

```
Qualifier      Status Bits (79-64)      Description
---------      -------------------      -------------------------
berr           0xxxx xx0x xxxx xxxy     Port bus error.
cpu            0x0xx xxxx xxxx 111xy     Function code CPU space.
data           0x0xx xxxx xxxx x01xy     Function code data space.
ds_byte        0xxxx xxx1 0xxx xxxxy     8-bit port.
ds_long        0xxxx xxx0 0xxx xxxxy     32-bit port.
ds_word        0xxxx xxx0 1xxx xxxxy     16-bit port.
fc3            0x1xx xxxx xxxx xxxxy     Function code FC3 line.
fetch          00xxx xxxx xxxx xxxxy     Instruction fetch cycle.
flush          0xxx0 xxxx xxxx xxxxy     First instruction following a pipeline flush.
halt           0xxxx x0xx xxxx xxxxy     Halt operation.
prog           0x0xx xxxx xxxx x10xy     Function code program space.
read           0xxxx xxxx xxx1 xxxxy     Read cycle.
retry          0xxxx x00x xxxx xxxxy     Retry of previous bus cycle.
siz_3byte      0xxxx xxxx x11x xxxxy     3 byte request.
siz_byte       0xxxx xxxx x01x xxxxy     Byte request.
siz_long       0xxxx xxxx x00x xxxxy     Long word request.
siz_word       0xxxx xxxx x10x xxxxy     Word request.
sup            0x0xx xxxx xxxx 1xxxy     Function code supervisor space.
supdata        0x0xx xxxx xxxx 101xy     Function code supervisor data space.
supprog        0x0xx xxxx xxxx 110xy     Function code supervisor program space.
user           0x0xx xxxx xxxx 0xxxy     Function code user space.
userdata       0x0xx xxxx xxxx 001xy     Function code user data space.
userprog       0x0xx xxxx xxxx 010xy     Function code user program space.
write          0xxxx xxxx xxx0 xxxxy     Write cycle.
xdma           0xx0x xxxx xxxx xxxxy     External DMA cycle.
```

Part 5

Installation Guide

Instructions for installing the product.

Part 5

15
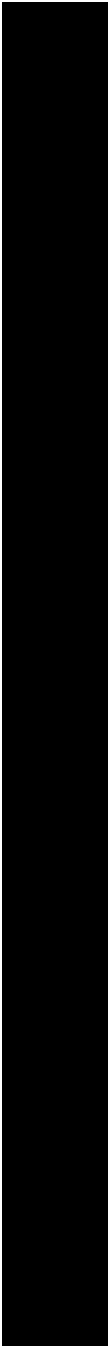
Installing the Debugger
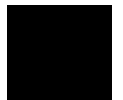
Installing the Debugger

This chapter shows you how to install the Real-Time C Debugger.

- Requirements

- If you are using the HP E3490A Software Probe

- Before Installing the Debugger

- Step 1. Connect the HP 64700 to the PC

- Step 2. Install the debugger software

- Step 3. Start the debugger

- Step 4. Check the HP 64700 system firmware version

- Optimizing PC Performance for the Debugger

Requirements

- IBM compatible or NEC PC with an 80486 microprocessor and 8 megabytes of memory.

- MS Windows 3.1, set up with 20 megabytes of swap space.

- VGA Display.

- 3 Megabytes available disk space.

If you are using an emulator:

- 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.

If you are using an HP E3490A Software Probe:

- Serial port.

- Novell LAN with Lan workplace for DOS or Microsoft Lan Manager with HP ARPA Services.
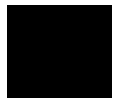
If You Are Using the HP E3490A Software Probe

1 Connect the HP E3490A Software Probe to the PC as described in the "HP E3490A Software Probe User's Guide" manual.

2 Skip to "Step 2. Install the debugger software."

Before Installing the Debugger

- Install MS Windows according to its installation manual. The Real-Time C Debugger must run under MS Windows in the 386 enhanced mode.

  To ensure your PC is running in the 386 Enhanced Mode, double-click the PIF Editor in the Main or Accessories window. Choose the Mode pulldown in the PIF Editor menu bar. A check mark should be beside "386 Enhanced" in the Mode pulldown.

- If the HP 64700 is to communicate with the PC via LAN:

  Make sure the HP 64700 LAN interface is installed (see the "HP 64700 Series Installation/Service" manual).

  Install the LAN card into the PC, and install the required PC networking software.

  Obtain the Internet Address, the Gateway Address, and the Subnet Mask to be used for the HP 64700 from your Network Administrator. These three addresses are entered in integer dot notation (for example, 192.35.12.6).

- If the HP 64700 is to communicate with the PC via RS-422:

  Install the HP 64037 RS-422 interface card into the PC. The Real-Time C Debugger includes software that configures the RS-422 interface.
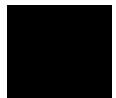
Step 1. Connect the HP 64700 to the PC

You can connect the HP 64700 to an RS-232 serial port on the PC, the Local Area Network that the PC is on, or an HP 64037 RS-422 interface that has been installed in the PC.

• To connect via RS-232

• To connect via LAN

• To connect via RS-422

To connect via RS-232

1 Set the HP 64700 configuration switches for RS-232C communication. Locate the COMM CONFIG switches on the HP 64700 rear panel, and set them as shown below.
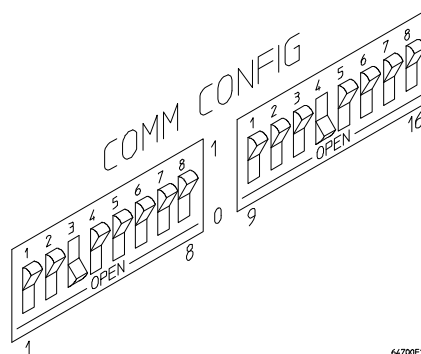
Notice that switches 1 through 3 are set to 001, respectively. This sets the baud rate to 19200.

Notice also that switches 12 and 13 are set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake which is needed to make sure all characters are processed.

2 Connect an RS-232C modem cable from the PC to the HP 64700 (for example, an HP 24542M 9-pin to 25-pin cable or an HP 13242N 25-pin to 25-pin cable).

If you want to build your own RS-232 cable, follow one of the pin-outs for HP cables shown in the following figure.
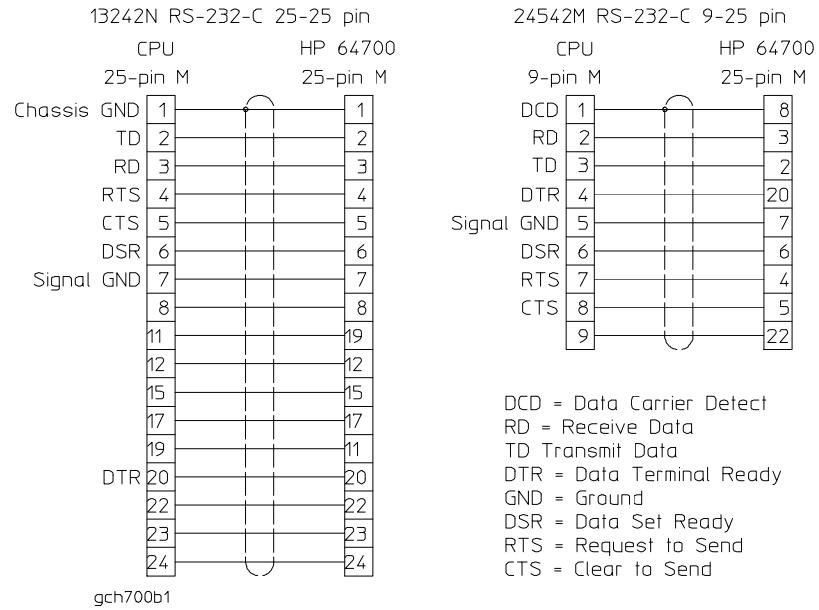
You can also use an RS-232C printer cable, but you must set HP 64700 configuration switch 4 to 1.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel. The power lamp at the lower right-hand corner of the front panel will light.

4 Start MS Windows in the 386 enhanced mode.

5 Verify RS-232 communication by using the Terminal program that is found in the Windows "Accessories" group box.

Double-click on the "Terminal" icon to open the Terminal window. Then, choose the Settings→Communications... (ALT, S, C) command, and select: 19200 Baud Rate, 8 Data Bits, 1 Stop Bit, Parity None, Hardware Flow Control, and the PC's RS-232 interface connector. Choose the OK button.

You should now be able to press the Enter key in the Terminal window to see the HP 64700's Terminal Interface prompt (for example, "R>", "M>", or "U>". The "->" prompt indicates the present firmware does not match the emulator probe, or there is no probe connected). If you see the prompt, you have verified RS-232 communication. If you do not see the prompt, refer to "If you cannot verify RS-232 communication".

If you will be using the RS-232 connection for the debugger, exit the Terminal program and go to "Step 2. Install the debugger software".

If you will be using the LAN connection, go to "To connect via LAN".

To connect via LAN

1 Set the HP 64700 LAN parameters.

If you're setting the HP 64700 LAN parameters for the first time, you must connect the HP 64700 to the PC via RS-232 before you can access the HP 64700 Terminal Interface. Follow the steps in "To connect via RS-232" and then return here.

If you're changing the LAN parameters of an HP 64700 that is already on the LAN, you can use the "telnet <HP 64700 IP address>" command to access the HP 64700 Terminal Interface.

Once the HP 64700 Terminal Interface has been accessed, display the current LAN parameters by entering the "lan" command:

```
R>lan
lan -i 15.6.25.117
lan -g 15.6.24.1
lan -s 255.255.248.0  <<- HP 64700A ONLY
lan -p 6470
Ethernet Address : 08000909BBC1
```

The "lan -i" line shows the Internet Address (or IP address). The Internet Address must be obtained from your Network Administrator. The value is entered in integer dot notation. For example, 192.35.12.6 is an Internet Address. You can change the Internet Address with the "lan -i <new IP>" command.

The "lan -g" line shows the Gateway Address which is also an Internet address and is entered in integer dot notation. This entry is optional and will default to 0.0.0.0, meaning all connections are to be made on the local network or subnet. If connections are to be made to workstations on other networks or subnets, this address must be set to the address of the gateway machine. The gateway address must be obtained from your Network Administrator. You can change the Gateway Address with the "lan -g <new gateway address>" command.

The "lan -s" line will be shown if you are using the HP 64700A, and will not be shown if you are using the HP 64700B. If this line is not shown, the Subnet Mask is automatically configured. If this line is shown, it shows the Subnet Mask in integer dot notation. This entry is optional and will default to 0.0.0.0. The default is valid only on networks that are not subnetted. (A network is

subnetted if the host portion of the Internet address is further partitioned into a subnet portion and a host portion.) If the network is subnetted, a subnet mask is required in order for the emulator to work correctly. The subnet mask should be set to all "1"s in the bits that correspond to the network and subnet portions of the Internet address and all "0"s for the host portion. The subnet mask must be obtained from your Network Administrator. You can change the Subnet Mask with the "lan -s <new subnet mask>" command .

Both the PC's subnet mask and the emulator's subnet mask must be identical unless they communicate via a gateway or a bridge. Unless your Network Administrator states otherwise, make them the same. You can check the PC's subnet mask with the "lminst" command if you are using HP-ARPA. If you are using Novell LAN WorkPlace, make sure the file \NET.CFG has the entry "ip_netmask <subnet mask>" in the section "Protocol TCPIP".

The "lan -p" line shows the base TCP service port number. The host computer interfaces communicate with the HP 64700 through two TCP service ports. The default base port number is 6470. The second port has the next higher number (default 6471). If the service port is not 6470, you must change it with the "lan -p 6470" command.

The Internet Address and any other LAN parameters you change are stored in nonvolatile memory and will take effect the next time the HP 64700 is powered off and back on again.

2 Exit the Terminal or telnet program.

3 Turn OFF power to the HP 64700.

4 Connect the HP 64700 to the LAN. This connection can be made using either the 15-pin AUI connector or the BNC connector.

DO NOT use both connectors. The LAN interface will not work with both connected at the same time.

5 Set the HP 64700 configuration switches for LAN communication.
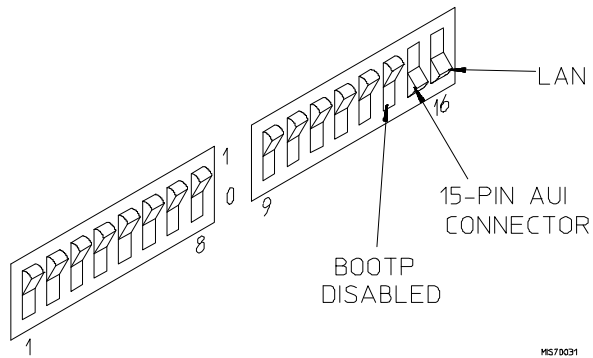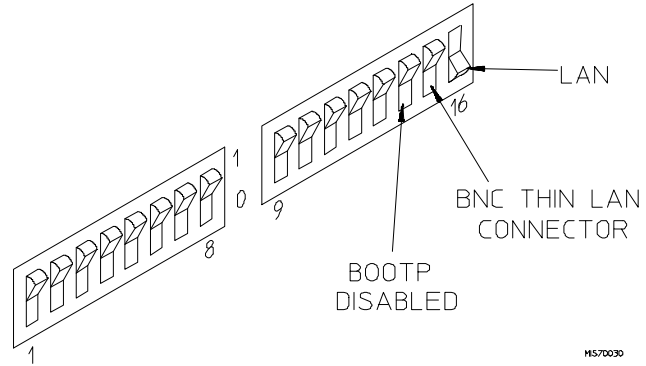
Switch 16 must be set to one (1) indicating that a LAN connection is being made.

Switch 15 should be zero (0) if you are connecting to the BNC connector or set to one (1) if a 15 pin AUI connection is made.

Switch 14 should be zero (0).

Set all other switches to zero (0).

6 Turn ON power to HP 64700.

7 Verify LAN communication by using a "telnet <HP 64700 IP address>" command. This connection will give you access to the HP 64700 Terminal Interface.

You should now be able to press the Enter key in the telnet window to see the HP 64700's Terminal Interface prompt (for example, "R>", "M>", "U>", etc.). If you see the prompt, you have verified LAN communication. If you cannot connect to the HP 64700's IP address, refer to "If you cannot verify LAN communication".

To connect via RS-422

Before you can connect the HP 64700 to the PC via RS-422, the HP 64037
RS-422 Interface must have already been installed into the PC.

1 Set the HP 64700 configuration switches for RS-422 communication.
Locate the COMM CONFIG switches on the HP 64700 rear panel, and
set them as shown below.
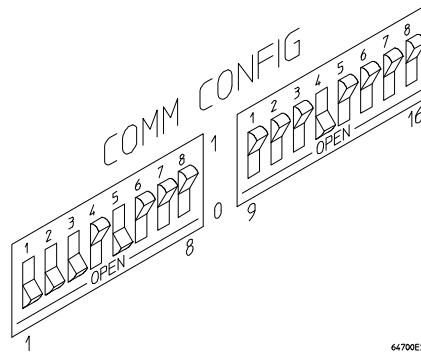
Notice that switches 1 through 3 are set to 111, respectively. This sets the
baud rate to 230400.

Notice that switch 5 is set to 1. This configures the 25-pin port for RS-422
communication.

Notice also that switches 12 and 13 are set to 1 and 0, respectively. This sets
the RTS/CTS hardware handshake which is needed to make sure all
characters are processed.

2 Connect the 17355M cable (which comes with the HP 64037
interface) from the PC to the HP 64700.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel.
The power lamp at the lower right-hand corner of the front panel will light.
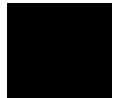
If you cannot verify RS-232 communication

If the HP 64700 Terminal Interface prompt does not appear in the Terminal window:

☐ Make sure that you have connected the emulator to the proper power source and that the power light is lit.

☐ Make sure that you have properly configured the data communications switches on the emulator and the data communications parameters on your controlling device. You should also verify that you are using the correct cable.

The most common type of data communications configuration problem involves the configuration of the HP 64700 as a DCE or DTE device and the selection of the RS-232 cable. If you are using the wrong type of cable for the device selected, no prompt will be displayed.

When the RS-232 port is configured as a DCE device (S4 is set to 0), a modem cable should be used to connect the HP 64700 to the host computer of terminal. Pins 2 and 3 at one end of a modem cable are tied to pins 2 and 3 at the other end of the cable.

When the RS-232 port is configured as a DTE device (S4 is set to 1), a printer cable should be used to connect the HP 64700 to the host computer of terminal. Pins 2 and 3 at one end of a printer cable are swapped and tied to pins 3 and 2, respectively, at the other end of the cable.

If you suspect that you may have the wrong type of cable, try changing the S4 setting and turning power to the HP 64700 OFF and then ON again.

If you cannot verify LAN communication

Use the "telnet" command on the host computer to verify LAN communication. After powering up the HP 64700, it takes a minute before the HP 64700 can be recognized on the network. After a minute, try the "telnet <internet address>" command.

• If "telnet" does not make the connection:

☐ Make sure that you have connected the emulator to the proper power source and that the power light is lit.

☐ Make sure that the LAN cable is connected. Refer to your LAN documentation for testing connectivity.

☐ Make sure the HP 64700 rear panel communication configuration switches are set correctly. Switch settings are only used to set communication parameters in the HP 64700 when power is turned OFF and then ON.

☐ Make sure that the HP 64700's Internet Address is set up correctly. You must use the RS-232 port to verify this that the Internet Address is set up correctly. While accessing the emulator via the RS-232 port, run performance verification on the HP 64700's LAN interface with the "lanpv" command.

• If "telnet" makes the connection, but no Terminal Interface prompt (for example, R>, M>, U>, etc.) is supplied:

☐ It's possible that the HP 64000 software is in the process of running a command (for example, if a repetitive command was initiated from telnet in another window). You can use CTRL+c to interrupt the repetitive command and get the Terminal Interface prompt.

☐ It's also possible for there to be a problem with the HP 64700 firmware while the LAN interface is still up and running. In this case, you must turn OFF power to the HP 64700 and turn it ON again.
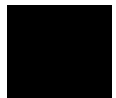
Step 2. Install the debugger software

1 If you are updating or re-installing the debugger software, you may want to save your b3627.INI file because it will be overwritten by the installation process.

2 Start MS Windows in the 386 enhanced mode.

3 Insert the 68360 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.

```
┌──────────────────────────────────────────────────┐
│ ▬                      Run                         │
├──────────────────────────────────────────────────┤
│  Command Line:                        ┌─────────┐  │
│  b:\setup                             │   OK    │  │
│                                       └─────────┘  │
│                                       ┌─────────┐  │
│  ☐ Run Minimized                      │ Cancel  │  │
│                                       └─────────┘  │
│                                       ┌─────────┐  │
│                                       │ Browse..│  │
│                                       └─────────┘  │
│                                       ┌─────────┐  │
│                                       │  Help   │  │
│                                       └─────────┘  │
└──────────────────────────────────────────────────┘
```
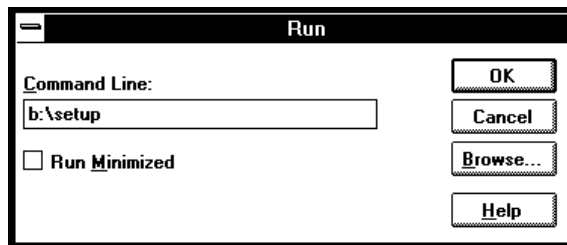
Then, choose the OK button. Follow the instructions on the screen.

You will be asked to enter the installation path. The default installation path is C:\HP\RTC\M360. The default installation path is shown wherever files are discussed in this manual.
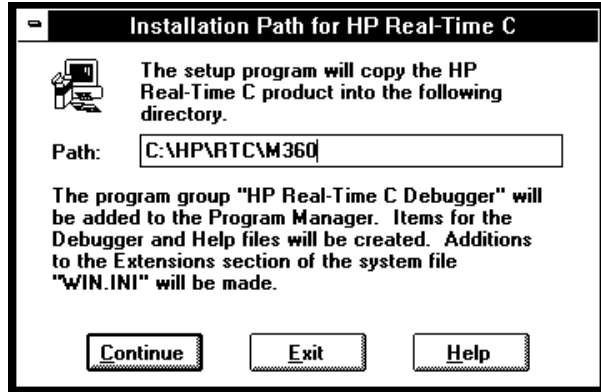
You will be asked to enter your user ID. This information is important if the HP 64700 is on the LAN and may be accessed by other users. It tells other users who is currently using, or who has locked, the HP 64700. This information can be modified while using the Real-Time C Debugger by choosing the Settings→Communication... (ALT, S, C) command.
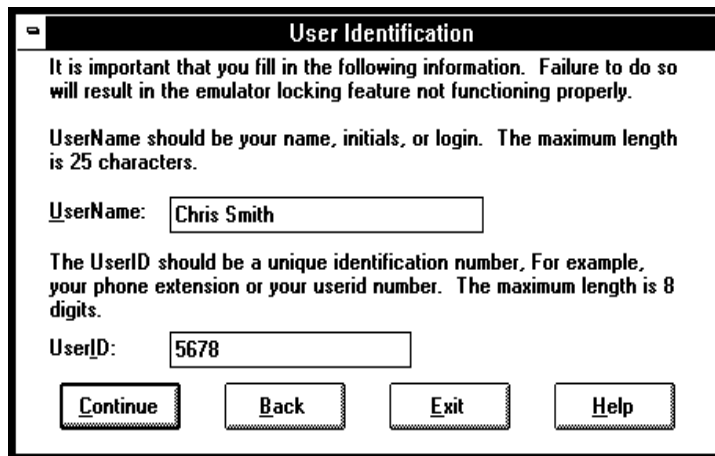
You will be asked to select the type of connection to be made to the HP 64700. This information can be modified while using the Real-Time C Debugger by choosing the Settings→Communication... (ALT, S, C) command.

```
┌──────────────────────────────────────────────────────┐
│ ▬        Real-Time C Emulation Connection             │
├──────────────────────────────────────────────────────┤
│                                                        │
│  Select the communication channel to be used between the │
│  personal computer and the emulator.                   │
│                                                        │
│  Transport:      │HP-ARPA              │  ▼            │
│                                                        │
│  Enter the connection name.  An example for RS232C would be │
│  COM1.  For HP-ARPA , use the emulator lan IP address or the │
│  emulator's network name.  For HP-RS422, use the address of │
│  the HP64037 card.                                     │
│                                                        │
│  Connection:     │15.6.25.24│                          │
│   ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐│
│   │ Continue │  │  Back    │  │  Exit    │  │  Help    ││
│   └──────────┘  └──────────┘  └──────────┘  └──────────┘│
└──────────────────────────────────────────────────────┘
```
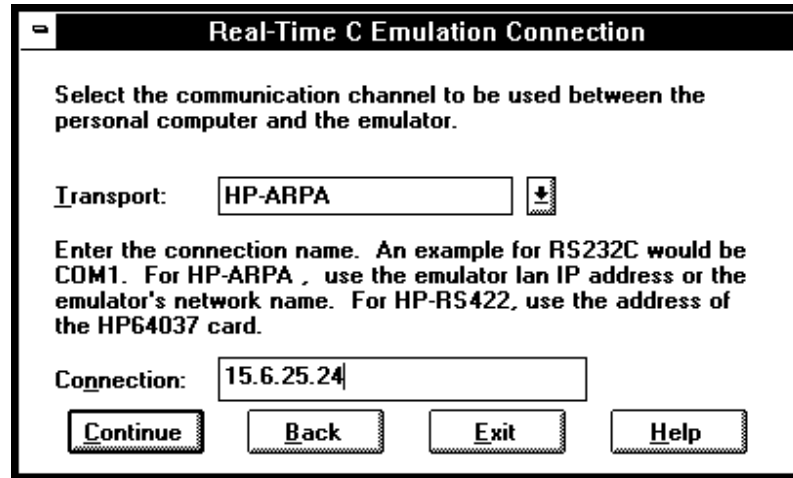
When using the HP-RS422 transport, the connection name is the I/O address you want to use for the HP 64037 card. Enter a hexadecimal number from 100H through 3F8H, ending in 0 or 8, that does not conflict with other cards in your PC.

After you have specified the type of connection, files will be copied to your hard disk. (The b3627.TMP and b3627.HLP files are larger than most of the other files and take longer to copy.) Fill out your registration information while waiting for the files to be copied.

If the Setup program detects that one or more of the files it needs to install are currently in use by Windows, a dialog box informs you that Windows must be restarted. You can either choose to restart Windows or not. If you don't choose to restart Windows, you can either run the _MSSETUP.BAT batch file (in the same directory that the debugger software is installed in) after you have exited Windows or reinstall the debugger software later when you are able to restart Windows.

Step 3. Start the debugger

1 If the "HP Real-Time C Debugger" group box is not opened, open it by double-clicking in the icon.

2 Double-click the "M68360 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 problems connecting to the HP E3490A Software Probe, refer to the "HP E3490A Software Probe User's Guide" manual.

If you have RS-232 connection problems

☐ Remember that Windows 3.1 only allows two active RS-232 connections at a time. To be warned when you violate this restriction, choose Always Warn in the Device Contention group box under 386 Enhanced in the Control Panel.

☐ Use the "Terminal" program (usually found in the Accessories windows program group) and set up the "Communications..." settings as follows:

```
Baud Rate: 19200 (or whatever you have chosen for the emulator)
Data Bits: 8
Parity: None
Flow Control: Hardware
Stop Bits: 1
```
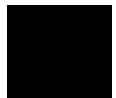
When you are connected, hit the Enter key. You should get a prompt back. If nothing echos back, check the switch settings on the back of the emulator.

Switches 1 thru 3 set the baud rate as follows:

```
S1  S2  S3
0   0   0   9600
0   0   1   19200
0   1   0   2400
```

Switches 12 and 13 must be set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake, which is needed to make sure all characters are processed.

All other switches should be in the "0" position, especially switch 16 on the HP 64700 (which selects LAN/Serial interface).

Remember that if you change any of the switch positions, you must turn OFF power to the HP 64700 and turn it ON again before the changes will take effect.

- If the switches are in the correct position and you still do not get a prompt when you press return, check the following:

☐ Turn off power to the HP 64700 and then turn it on again. Press return to see if you get a prompt.

☐ Check to make sure the RS-232 cable is connected to the correct port on your PC, and that the cable is appropriate for connecting the PC to a DCE device. If the cable is intended to connect the PC to a DTE device, set switch 4 to "1" (which makes the emulator a DTE device), turn OFF power to the HP 64700, turn power ON, and try again.

☐ Check to make sure your RS-232 cable has the RTS, CTS, DSR, DCD, and DTR pins supported. If your PC RS-232 connection is a 9-pin male connection, HP cable number 24542M will work (set switch 4 to 0 if you use this cable). If your PC has a 25-pin RS-232 connector, HP cable number 13242N will work (set switch 4 to 0).

- If you wish to build your own RS-232 cable, refer to "To connect via RS-232" in the paragraph titled, "Step 1. Connect the HP 64000 to the PC" earlier in this chapter.

- When using certain RS-232 cards, connecting to an RS-232 port where the HP 64700 is turned OFF (or not connected) will halt operation of the PC. The only way to restore operation is to reboot the PC. Therefore, HP recommends you always turn ON the HP 64700 before attempting to connect via RS-232.

- If RTC reports overrun errors or simply times out, RTC may be overrunning the serial interface. In this case, try the following:

☐ Stop all unnecessary TSR's and other applications to allow the processor to service the serial interface more often.

☐ Overrun errors may occur when the serial interface card is not sufficiently buffered. Check to make sure your serial interface card uses the 16550AF UART, or better. Use the DOS command, "MSD", and when the window opens, select "COM Ports..." to see the UART chip used in your serial interface card.
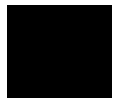
If you have LAN connection problems

☐ Try to "ping" the emulator:

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

☐ If the emulator does not respond:

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

- Check that switch 15 on the emulator is in the correct position for your LAN interface (either the AUI or the BNC).

Remember, if you change any switch settings on the emulator, the changes do not take effect until you turn OFF emulator power and turn it ON again.

☐ If the emulator still does not respond to a "ping," you need to verify the IP address and subnet mask of the HP 64700. To do this, connect the HP 64700 to a terminal (or to the Terminal application on the PC), change the emulator's switch settings so it is connected to RS-232, and enter the "lan" command. The output looks something like this:

```
lan -i 15.6.25.117
lan -g 15.6.24.1
lan -s 255.255.248.0
lan -p 6470
Ethernet Address : 08000909BBC1
```

The important outputs (as far as connecting) are:

"lan -i"; this shows the internet address is 15.6.25.117 in this case. If the Internet address (IP) is not what you expect, you can change it with the 'lan -i <new IP>' command.

"lan -s"; shows the subnet mask is 255.255.248 (the upper 21 bits -- 255.255.248.0 == FF.FF.F8.0). If the subnet mask is not what you expect, you can change it with the 'lan -s <new subnet mask>' command.

"lan -p"; shows the port is 6470. If the port is not 6470, you must change it with the "lan -p 6470" command.

Both the PC's subnet mask and the emulator's subnet mask must be identical unless they communicate via a gateway or a bridge. Unless your Network

Administrator states otherwise, make them the same. If you are using HP-ARPA, you can check the PC's subnet mask with the "lminst" command in a DOS window. If you are using Novell LAN WorkPlace, make sure the file \\NET.CFG has the entry "ip_netmask <subnet mask>" in the section "Protocol TCPIP." If you are using Windows for Workgroups, you can check the PC's subnet mask by looking in the [TCPIP] section of the PROTOCOL.INI file or by looking in the Microsoft TCP/IP Configuration dialog box. If you are using WINSOCK, refer to your LAN software documentation for subnet mask information.

☐ Occasionally the emulator or the PC will "lock up" the LAN due to excessive network traffic. If this happens, all you can do is turn OFF power to the HP 64700 or PC and turn it back ON, again. If this happens two frequently, you can try placing a gateway between the emulator/PC and the rest of your network.

If you have LAN DLL errors

The various LAN transport selections require the following DLLs:

HP-ARPA　　　　WSOCKETS.DLL.

Novell-WP　　　　WLIBSOCK.DLL.

W4WG-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.
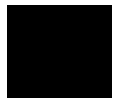
If you have RS-422 connection problems

☐ 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, especially the switch that determines LAN/Serial interface (switch 16 on HP 64700).

Remember that if you change any of the switch positions, you must turn OFF power to the HP 64700 and turn it ON again before the changes will take effect.

☐ If the switches are in the correct position and you still do not get a prompt when you hit return, try turning OFF the power to the HP 64700 and tuning it ON again.

☐ If you still don't get a prompt, make sure the HP 17355M RS-422 cable is connected to the correct port on your PC.

Step 4. Check the HP 64700 system firmware version

- Choose the Help→About Debugger/Emulator... (ALT, H, D) command.

The version information under HP 64700 Series Emulation System must show A.04.00 or greater. If the version number is less than A.04.00, you must update your HP 64700 system firmware as described in the Installing/Updating HP 64700 Firmware chapter.
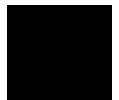
Optimizing PC Performance for the Debugger

The Real-Time C Debugger is a memory and I/O intensive Windows program. Slow user interface performance may be caused by many things:

- Underpowered PC -- The Real-Time C Debugger requires an IBM compatible or NEC PC with an 80486 class microprocessor, 8 megabytes of memory, and 20 megabytes of MS Windows swap space. Because RAM is faster than swap, performance is best when there is enough RAM to accommodate all of the Real-Time C Debugger's memory usage (which is directly related to the size of your programs and the amount of debug information in them).

- Improperly configured PC -- Windows configuration may have a very significant effect on performance. The Windows swap file settings are very important (see the Virtual Memory dialog box under 386 Enhanced in the Control Panel). The larger the swap file, the better the performance.   Permanent swap has superior performance.

- Disk performance (due to Windows swap file access and Windows dialog and string resource accesses from the debugger ".EXE" file) -- The disk speed has a direct impact on performance of the Real-Time C Debugger. Use of SMARTDrive or other RAM disk or caching software will improve the performance.

Various PC performance measurement and tuning tools are commercially available. Optimizing your PC performance will improve debugger interface performance and, of course, all your other PC applications will benefit as well.

16

Installing/Updating HP 64700
Firmware

Installing/Updating HP 64700 Firmware

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

If you are using an HP 64700A, it must contain the optional Flash EPROM memory card before you can install or update HP 64700 system firmware. Flash EPROM memory is standard in the HP 64700B card cage.

The firmware, and the program that downloads it into the HP 64700, are included with the debugger on floppy disks labeled HP 64700 EMUL/ANLY FIRMWARE.

The steps to install or update HP 64700 firmware are:

- Step 1. Connect the HP 64700 to your PC

- Step 2. Install the firmware update utility

- Step 3. Run PROGFLASH to update HP 64700 firmware

- Step 4. Verify emulator performance

Step 1. Connect the HP 64700 to the PC

1 Set the COMM CONFIG switches for RS-232C communication. To do this, locate the DIP switches on the HP 64700 rear panel, and set them as shown below.
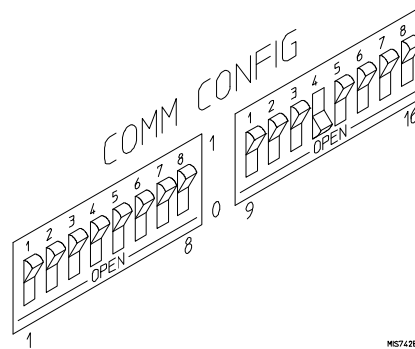
Notice that switches 12 and 13 are set to 1 and 0, respectively. This sets the RTS/CTS hardware handshake, which is needed to make sure all characters are processed. Switches 1, 2, and 3 are set to 0. This sets the baud rate to 9600. Switch settings are read during the HP 64700 power up routine.

2 Connect an RS-232C modem cable from the PC to the HP 64700 (for example, an HP 24542M 9-pin to 25-pin cable or an HP 13242N 25-pin to 25-pin cable).

You can also use an RS-232C printer cable, but if you do, you MUST set COMM CONFIG switch 4 to 1.

3 Turn ON power to the HP 64700.

The power switch is located on the lower left-hand corner of the front panel. The power lamp at the lower right-hand corner of the front panel will light.

4 Start MS Windows in the 386 enhanced mode.

To ensure your PC is running in the 386 Enhanced Mode, double-click the PIF Editor in the Main or Accessories window. Choose the Mode pulldown in the PIF Editor menu bar. A check mark should be beside "386 Enhanced" in the Mode pulldown.

5 Verify RS-232 communication by using the Terminal program that is found in the Windows "Accessories" group box.

Double-click on the "Terminal" icon to open the Terminal window. Then, choose the Settings→Communications... (ALT, S, C) command, and select: 9600 Baud Rate, 8 Data Bits, 1 Stop Bit, Parity None, Hardware Flow Control, and the PC's RS-232 interface connector to which the RS-232 cable is attached (example: COM1). Choose the OK button.

You should now be able to press the Enter key in the Terminal window to see the HP 64700's Terminal Interface prompt (for example, p>, R>, M>, and U>. A -> prompt indicates the present firmware does not match the emulator probe, or there is no probe connected). If you see the prompt, you have verified RS-232 communication. If you do not see the prompt, refer to "If you cannot verify RS-232 communication" in Chapter 15.

6 Exit the Terminal window.

Step 2. Install the firmware update utility

The firmware update utility and emulation and analysis firmware require about 1.5 Mbytes of disk space.

1 Start MS Windows in the 386 enhanced mode.

2 Insert the HP 64700 EMUL/ANLY FIRMWARE Disk 1 of 2 into floppy disk drive A or B.

3 Choose the File→Run... (ALT, F, R) command in the Windows Program Manager. Enter "a:\setup" (or "b:\setup" if you installed the floppy disk into drive B) in the Command Line text box.
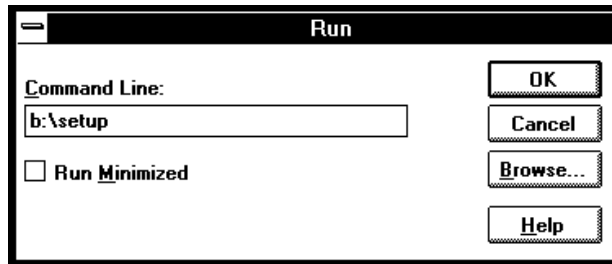
Then, choose the OK button. Follow the instructions on the screen.

You will be asked to enter the installation path. The default installation path is C:\HP64700.
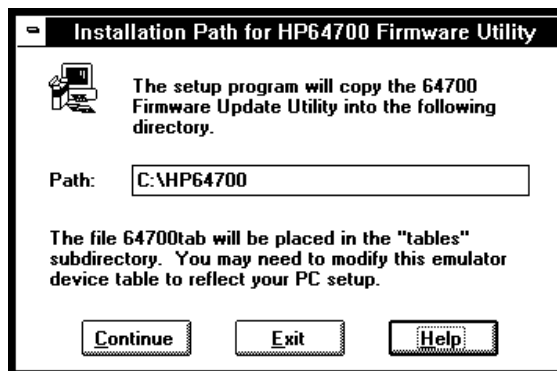
Wait until the Setup Exit Message dialog box appears. This indicates installation of the firmware update utility is complete.

4 After completing the installation, use the editor of your choice and edit the C:\CONFIG.SYS file to include these lines:

```
BREAK=ON
FILES=20
```

BREAK=ON allows the system to check for two break conditions: CTRL+Break, and CTRL+c.

FILES=20 allows 20 files to be accessed concurrently. This number must be at LEAST 20 to allow the firmware update utility to operate properly.

5 If you installed the files in a path other than the default (C:\HP64700), edit the C:\AUTOEXEC.BAT and C:\HP64700\BIN\FLASH.BAT files as follows:

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

```
SET HP64700=C:\<installation_path>
SET HPTABLES=C:\<installation_path>\TABLES
```

• Edit FLASH.BAT to identify the location of PROGFLAS.EXE. For example:

```
C:\<installation_path>\PROGFLAS.EXE
```

6 Edit the <installation_path>\TABLES\64700TAB file to indicate the communications connection you will use, as follows:

The default <installation_path>\TABLES\64700TAB file contains entries to establish the communications connection for COM1 and COM2. The content of this file is:

```
EMUL_COM1 unknown COM1 OFF 9600 NONE ON 1 8
EMUL_COM2 unknown COM2 OFF 9600 NONE ON 1 8
```

If you are using COM3 or COM4 port to update your firmware, you need to edit the <installation_path>\TABLES\64700TAB file. Either add another line or modify one of the existing lines. For example:

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

7 Ensure the Interrupt Request Line for the selected COMx port is set to its default value. To check the default value:

    1 Choose Control Panel in the Main window.

    2 Choose Ports in the Control Panel window.

    3 Choose the COMx port you are using and click Settings....

    4 Click Advanced... in the Settings for COMx dialog box.

    5 Select the default value for the Interrupt Request Line in the Advanced Settings for COMx dialog box. The default settings are:

```
COM1 and COM3 = IRQ 4
COM2 and COM4 = IRQ 3
```

8 Exit Windows and reboot your PC to activate the changes made to the CONFIG.SYS and AUTOEXEC.BAT files (CTRL+ALT+DEL). Installation of the firmware update utility is now complete.

Step 3. Run PROGFLASH to update HP 64700 firmware

1 Start MS Windows in the 386 enhanced mode.

2 If the "HP 64700 Firmware Utility" group box is not opened, open it by double-clicking the icon.

3 Double-click the "PROGFLASH" icon. (You can abort the PROGFLASH command by pressing CTRL+c.)

4 Enter the number that identifies the emulator you want to update. For example, enter "1" if you want to update the emulator identified by the line, "1 emul_com1 my_emul."

5 Enter the number that identifies the product whose firmware you want to update. For example, if this product is listed as number 12, enter "12":

```
Product
 1  64782
 2  E3490
 .
 .
 12  647??
 .
```

6 Enter "y" to enable status messages.

The PROGFLASH command downloads code from files on the host computer into Flash EPROM memory in the HP 64700. During this download, you will see messages similar to the following:

```
Rebooting HP64700...with init -r

Downloading flash programming code:
'/hp64700/lib/npf.X'
Checking Hardware id code...
Erasing Flash ROM
Downloading ROM code: '/hp64700/update/647??.X'
  Code start 280000H
  Code size 29ABAH
Finishing up...

Rebooting HP64700...
Flash programming SUCCEEDED
```

You can display firmware version information and verify the update by choosing the Help→About Debugger/Emulator... (ALT, H, D) command in the Real-Time C Debugger.

Step 4. Verify emulator performance

- Do the performance verification procedure shown in the
  Installation/Service/Terminal Interface User's Guide.

Glossary

Defines terms that are used in the debugger help information.

analyzer An instrument that captures data on signals of interest at discreet periods. The emulation bus analyzer captures emulator bus cycle information synchronously with the processor's clock signal.

arm condition A condition that enables the analyzer. The analyzer is always armed unless you set the analyzer up to be armed by a signal received on the BNC port; when you do this, you can identify the arm condition in the trace specification by selecting arm in the Condition dialog boxes.

background memory A separate memory system, internal to the emulator, out of which the background monitor executes.

background monitor program An emulation monitor program that executes out of background memory.

break on trigger Causes emulator execution to break into the monitor when the trigger condition is found. This is known as a hardware breakpoint, and it lets you break on a wider variety of conditions than a software breakpoint (which replaces an opcode with a break instruction); however, depending on the speed of the processor, the actual break point may be several cycles after the one that caused the trigger.

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

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

control menu The menu that is accessed by clicking the control menu box in the upper left corner of a window. You can also access control menus by pressing the "ALT" and "-" keys.

count condition Specifies whether time or the occurrences of a particular state are counted for each state in the trace buffer.

dual-port emulation memory Emulation memory that can be displayed or modified while programs run in real time.

embedded microprocessor system The microprocessor system that the emulator plugs into.

emulation memory Memory provided by the emulator that can be used in place of memory in the target system.

emulation monitor A program, executed by the emulation microprocessor (as directed by the emulation system controller), that gives the emulator access to target system memory, microprocessor registers, and other target system resources.

emulator An instrument that performs just like the microprocessor it replaces, but at the same time, it gives you information about the operation of the processor. An emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory, and I/O resources.

enable condition Specifies the first condition in a two-step sequential trigger condition.

enable store condition Specifies which states get stored in the trace buffer while the analyzer searches for the enable condition.

foreground memory The memory system out of which user programs execute. Foreground memory is made up of emulation memory and target system memory.

foreground monitor program An emulation monitor program that executes out of the same memory system as user programs. This memory system is known as foreground memory and is made up of emulation memory and target system memory. The emulator only allows foreground monitor programs in emulation memory.

function code Identifies 68360 memory space:

X	Same as no function code.
S	Specifies "Supervisor" memory space.
U	Specifies "User" memory space.
P	Specifies "Program" memory space.
D	Specifies "Data" Memory space.
SP	Specifies "Supervisor Program" memory space.
SD	Specifies "Supervisor Program" memory space.
UP	Specifies "User Program" memory space.
UD	Specifies "User Data" memory Space.
CPU	Specifies "CPU" space.

guarded memory Memory locations that should not be accessed by user programs. These locations are specified when mapping memory. If the user program accesses a location mapped as guarded memory, emulator execution breaks into the monitor.

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

monitor A program, executed by the emulation microprocessor (as directed by the emulation system controller), that gives the emulator access to target system memory, microprocessor registers, and other target system resources.

object file An Intel OMF format absolute file that can be loaded into emulation or target system memory and executed by the debugger.

pop-up menu A menu that is accessed by clicking the right mouse button in a window.

prestore condition Specifies the states that may be stored before each normally stored state. Up to two states may be prestored for each normally stored state.

primary branch condition Specifies a condition that causes the analyzer to begin searching at another level.

restart condition Specifies the condition that restarts the two-step sequential trigger. In other words, if the restart condition occurs while the analyzer is searching for the trigger condition, the analyzer starts looking for the enable condition again.

secondary branch condition Specifies a condition that causes the analyzer to begin searching at another level. If a state satisfies both the primary and secondary branch conditions, the primary branch will be taken.

sequence levels Levels in the analyzer that let you specify a complex sequential trigger condition. For each level, the analyzer searches for primary and secondary branch conditions. You can specify a different store condition for each level. The Page button toggles the display between sequence levels 1 through 4 and sequence levels 5 through 8.

state qualifier A combination of address, data, and status values that identifies particular states captured by the analyzer.

status values Values that identify the types of microprocessor bus cycles recognized by the analyzer. You can include status values (along with address and data values) when specifying trigger and store conditions. The status values defined for the 68360 emulator are listed under "Predefined Status Values" at the end of Chapter 14, "Concepts."

store condition Specifies which states get stored in the trace buffer.

In the "Find Then Trigger" trace set up, the store condition specifies the states that get stored after the trigger.

In the "Sequence" trace set up, each sequence level has a store condition that specifies the states that get stored while looking for the primary or secondary branch conditions.

target system The microprocessor system that the emulator plugs into.

trace state The information captured by the analyzer on a particular microprocessor bus cycle.

transfer address The program's starting address defined by the software development tools and included with the symbolic information in the object file.

trigger The captured analyzer state about which other captured states are stored. The trigger state specifies when the trace measurement is taken.

trigger condition Specifies the condition that causes states to be stored in the trace buffer.

trigger position Specifies whether the state that triggered the analyzer appear at the start, center, or end of the trace buffer. In other words, the trigger position specifies whether states are stored after, about, or before the trigger.

trigger store condition Specifies which states get stored in the trace buffer while the analyzer searches for the trigger condition.

watchpoint A variable that has been placed in the WatchPoint window where its contents can be readily displayed and modified.

Index

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING   Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).

Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.

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.