

***TMS370 Family  
Simulator  
Getting Started Guide***



## **IMPORTANT NOTICE**

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

**TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.**

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

## Preface

# Read This First

---

---

---

### ***About This Manual***

This manual tells you how to install the simulator version of the C source debugger on a PC™ running Windows™. This manual also tells you how to invoke the simulator version of the debugger and introduces the simulator's basic features.

### ***Notational Conventions***

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a special *typeface* similar to a typewriter's. Examples use a **bold version** of the special typeface for emphasis; interactive displays use a **bold version** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

Here is an example of a system prompt and a command that you might enter:

```
C: MD C:\370HLL
```

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered. Here is an example of a command syntax:

```
pinc pinname, filename
```

**pinc** is the command. This command has two parameters, indicated by *pinname* and *filename*.

- Square brackets ( [ and ] ) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you don't enter the brackets themselves. Here's an example of a command that has an optional parameter:

```
sim370w [filename] [options]
```

The **sim370w** command has two parameters. The both parameters are optional.

- Braces ( { and } ) indicate a list. The symbol | (read as *or*) separates items within the list. Here's an example of a command including a list:

```
mc portaddress, length, filename, {READ | WRITE}
```

This provides two choices: READ or WRITE.

Unless the list is enclosed in square brackets, you must choose one item from the list.

### **Related Documentation From Texas Instruments**

The following books describe the TMS370 and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

***TMS370 and TMS370C8 8-Bit Microcontroller Family Assembly Language Tools User's Guide*** (literature number SPNU010) describes the assembly language tools (assembler, linker, and other tools used to develop assembly code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS370/C8 8-bit family of devices.

***TMS370 and TMS370C8 8-Bit Microcontroller Family Optimizing C Compiler User's Guide*** (literature number SPNU022) describes the TMS370/C8 8-bit C compiler. This C compiler accepts ANSI standard C source code and produces assembly language source code for the TMS370/C8 8-bit family of devices.

***TMS370 Family C Source Debugger User's Guide*** (literature number SPNU028) tells you how to invoke the '370 XDS/22 emulator and application board versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints, and includes a tutorial that introduces basic debugger functionality. It also includes an advanced tutorial that introduces the breakpoint, trace, and timing features.

***TMS370 Microcontroller Family User's Guide*** (literature number SPNU127) discusses hardware aspects of the TMS370 family members, such as pin functions, architecture, module options, stack operation, and interfaces. The manual also contains the TMS370 assembly language instruction set.

## Trademarks

Microsoft and Windows are registered trademarks of Microsoft Corporation.

PC is a trademark of International Business Machines Corporation.

Pentium is a trademark of Intel Corporation.

XDS/22 is a trademark of Texas Instruments Incorporated.

## If You Need Assistance . . .

If you want to . . .	Contact Texas Instruments at . . .
Visit TI online	World Wide Web: <a href="http://www.ti.com">http://www.ti.com</a>
Receive general information or assistance	World Wide Web: <a href="http://www.ti.com/sc/docs/pic/home.htm">http://www.ti.com/sc/docs/pic/home.htm</a> North America, South America: (214) 644-5580 Europe, Middle East, Africa Dutch: 33-1-3070-1166 English: 33-1-3070-1165 French: 33-1-3070-1164 Italian: 33-1-3070-1167 German: 33-1-3070-1168 Japan (Japanese or English) Domestic toll-free: 0120-81-0026 International: 81-3-3457-0972 or 81-3-3457-0976 Korea (Korean or English): 82-2-551-2804 Taiwan (Chinese or English): 886-2-3771450
Ask questions about micro-controller product operation or report suspected problems	Hotline: (713) 274-2370 Fax: (713) 274-4203 Email: *H370@msg.ti.com World Wide Web: <a href="http://www.ti.com/sc/micro">http://www.ti.com/sc/micro</a> BBS: (713) 274-3700 8-N-1
Request tool updates	Software: (214) 638-0333 Software fax: (214) 638-7742 Hardware: (713) 274-2285
Order Texas Instruments documentation (see Note 1)	Literature Response Center: (800) 477-8924
Make suggestions about or report errors in documentation (see Note 2)	Email: <a href="mailto:comments@books.sc.ti.com">comments@books.sc.ti.com</a> Mail: Texas Instruments Incorporated Technical Publications Manager, MS 702 P.O. Box 1443 Houston, Texas 77251-1443

- Notes:**
- 1) The literature number for the book is required; see the lower-right corner on the back cover.
  - 2) Please mention the full title of the book, the literature number from the lower-right corner of the back cover, and the publication date from the spine or front cover.



# Contents

---

---

---

<b>1</b>	<b>Installing the Simulator With Windows</b> .....	<b>1-1</b>
	<i>Lists the hardware and software you'll need to install the simulator version of the C source debugger; provides installation instructions for PC systems running Windows.</i>	
1.1	System Requirements .....	1-2
	Hardware checklist .....	1-2
	Software checklist .....	1-3
1.2	Step 1: Installing the Debugger Software .....	1-4
	Creating a program group .....	1-4
	Using a program-item icon .....	1-5
1.3	Step 2: Setting Up the Debugger Environment .....	1-6
	Modifying the PATH statement .....	1-7
	Setting up the environment variables .....	1-7
	Invoking the new or modified batch file .....	1-8
1.4	Step 3: Verifying the Installation .....	1-9
<b>2</b>	<b>Simulator Features</b> .....	<b>2-1</b>
	<i>Describes the simulator-specific features of the C source debugger.</i>	
2.1	Invoking the Debugger .....	2-2
	Selecting the screen size (-b and -bb options) .....	2-3
	Identifying additional directories (-i option) .....	2-3
	Selecting the minimal debugging mode (-min option) .....	2-3
	Entering the profiling environment (-profile option) .....	2-4
	Loading the symbol table only (-s option) .....	2-4
	Identifying a new initialization file (-t option) .....	2-4
	Loading without the symbol table (-v option) .....	2-4
	Ignoring D_OPTIONS (-x option) .....	2-4
2.2	A Sample Memory Map for the Simulator .....	2-5

2.3	Identifying Usable Memory Ranges .....	2-6
	Restrictions on usable memory ranges .....	2-8
2.4	Simulating I/O Space .....	2-9
	Connecting a peripheral I/O port .....	2-9
	Disconnecting a peripheral I/O port .....	2-10
2.5	Simulating Interrupts .....	2-11
	Setting up your input file .....	2-11
	Programming the simulator .....	2-12
2.6	Using Predefined Constants With Conditional Commands .....	2-14
2.7	Benchmarking .....	2-15
2.8	Profiling Code Execution .....	2-15
2.9	Debugger Messages .....	2-16



# Installing the Simulator With Windows

---

---

---

---

This chapter helps you install the simulator version of the C source debugger on a PC running Windows. The debugger is the programmer's interface to the TMS370 family simulator. When you complete the installation, turn to Chapter 2, *Simulator Features*, for more information about using the simulator version of the '370 debugger.

<b>Topic</b>	<b>Page</b>
<b>1.1 System Requirements</b> .....	<b>1-2</b>
<b>1.2 Step 1: Installing the Debugger Software</b> .....	<b>1-4</b>
<b>1.3 Step 2: Setting Up the Debugger Environment</b> .....	<b>1-6</b>
<b>1.4 Step 3: Verifying the Installation</b> .....	<b>1-9</b>

## 1.1 System Requirements

To install and use the '370 family C source debugger, you need the items in the following hardware and software checklists.

### **Hardware checklist**

- |                          |                                |   |
|--------------------------|--------------------------------|---|
| <input type="checkbox"/> | <b>Host</b>                    | 32-bit x86-based or Pentium™ PC with a hard-disk system and a 1.44-Mbyte floppy-disk drive  |
| <input type="checkbox"/> | <b>Memory</b>                  | Minimum of 4 Mbytes of RAM  |
| <input type="checkbox"/> | <b>Display</b>                 | Monochrome or color monitor (color recommended)   |
| <input type="checkbox"/> | <b>Optional hardware</b>       | Microsoft™-compatible mouse   |
| <input type="checkbox"/> |                                | EGA- or VGA-compatible graphics display card and a large (17" or 19") monitor. The debugger has two options that allow you to enlarge the overall size of the debugger display. To use a larger screen size, you must invoke the debugger with an appropriate option. For more information, see the <i>Selecting the screen size (-b and -bb options)</i> discussion on page 2-3. |
| <input type="checkbox"/> | <b>Miscellaneous materials</b> | Blank, formatted disks  |

**Software checklist**

- Operating system** Windows version 3.1 or later
- Software tools** TMS370 8-bit microcontroller family assembler and linker
- Optional: TMS370 8-bit microcontroller family C compiler
- Optional files included with the debugger package**

*init.cmd* is a general-purpose batch file that contains debugger commands. This batch file, shipped with the debugger, defines a '370 memory map. When you start using the debugger, this memory map should be sufficient for your needs. Later, you may want to define your own memory map. For information about setting up your own memory map, see the *TMS370 Family C Source Debugger User's Guide*.
- init.clr* is a general-purpose screen configuration file. If *init.clr* isn't present when you invoke the debugger, the debugger uses the default screen configuration.
- Several *.clr* (for color monitors) and *.mon* (for monochrome monitors) screen configuration files are included in the *screens* directory. When you first invoke the debugger, the default screen configuration should be sufficient for your needs. Later, you may want to define your own custom configuration.

For information about these files and about setting up your own screen configuration, see the *TMS370 Family C Source Debugger User's Guide*.

## 1.2 Step 1: Installing the Debugger Software

This section explains the process of installing the simulator version of the debugger on a hard-disk system.

- 1) Make a backup copy of the debugger product disk.
- 2) On your hard disk or system disk, create a directory named *370hll*. This directory will contain the '370 C source debugger software. To create this directory, enter:

```
MD C:\370HLL
```

- 3) Insert the debugger product disk into drive A. Copy the contents of the disk:

```
XCOPY /V /S A:\*.* C:\370HLL
```

You may want to create a Windows program group and use a program-item icon to make it easier to invoke the debugger from within the Windows environment.

### **Creating a program group**

A program group contains program-item icons. You can use program groups to help you organize your icons. To create a program group, follow these steps:

- 1) From the Windows Program Manager, select File → New. This displays the New Program Object dialog box.
- 2) Make sure Program Group is selected.
- 3) Click on OK. This displays the Program Group Properties dialog box.
- 4) Enter a name for the program group in the Description field.
- 5) Click on OK. This displays an empty program group with the name you entered.

### **Using a program-item icon**

A program-item icon represents an application that you can run from Windows. Program-item icons are contained inside program groups. The debugger already has a standard icon that you can use. To use the standard debugger icon, follow these steps:

- 1) If the program group in which you want to place the icon is not already open, double-click on it to open it.
- 2) Open the Windows File Manager.
- 3) Arrange the windows so you can see the program group and the File Manager at the same time.
- 4) In the 370hll directory of the Windows File Manager, click once on sim370w.exe to select the executable file.
- 5) Drag and drop the sim370w.exe into the program group. An icon that looks like this displays:



You can now close the File Manager.

- 6) Click once on the program icon to select it.
- 7) In the Program Manager, select File → Properties. This displays the Program Item Properties dialog box.
- 8) Modify the information in the Command Line field to include the options you normally use at start-up. For a summary of the options that you can use, see Table 2–1, *Summary of Debugger Options*, on page 2-2.

### 1.3 Step 2: Setting Up the Debugger Environment

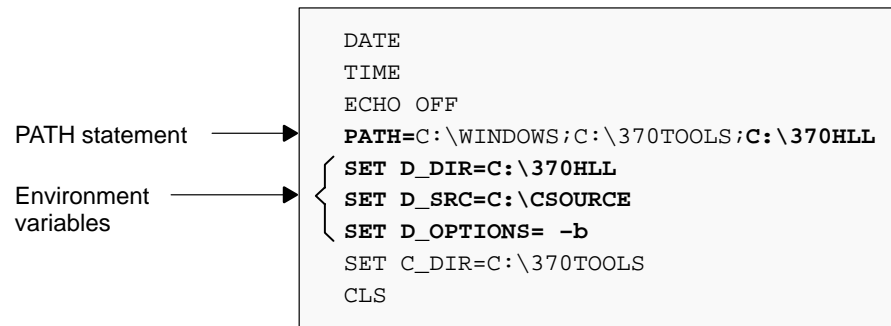
To ensure that your debugger works correctly, you must perform some tasks before you invoke the debugger for the first time or anytime you power up or reboot your PC. You can perform these tasks by entering individual DOS commands, but it is simpler to put the commands in a batch file. You can edit your system's autoexec.bat file, but in some cases, modifying that file can interfere with other applications running on your PC. You can create a separate batch file to perform these tasks instead. No matter which way you choose to do them, these are the tasks you must perform:

- Modify the PATH statement to identify the 370hll directory.
- Define environment variables so that the debugger can find the files it needs.

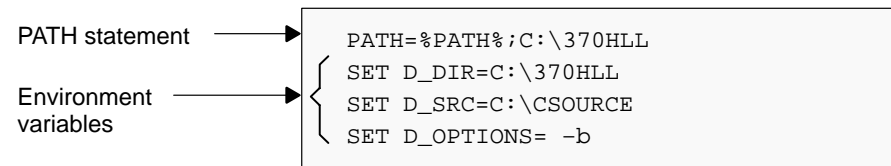
Figure 1–1 (a) shows an example of an autoexec.bat file that contains the suggested modifications. Figure 1–1 (b) shows a sample batch file that you could create instead of editing the autoexec.bat file. The subsections following the figure explain these modifications.

Figure 1–1. DOS-Command Setup for the Debugger

(a) Sample autoexec.bat file to use with the debugger



(b) Sample batch file that you create to use with the debugger



### **Modifying the PATH statement**

Define a path to the debugger directory. The general format for doing this is:

```
PATH=C:\370HLL
```

This allows you to invoke the debugger without specifying the name of the directory that contains the debugger executable file.

- If you are modifying an autoexec.bat file that already contains a PATH statement, simply include ;C:\370hll at the end of the statement, as shown in Figure 1-1 (a).
- If you are creating your own batch file, use a different format for the PATH statement:

```
PATH=%PATH%;C:\370HLL
```

The addition of %**path**%; ensures that this PATH statement won't undo PATH statements in any other batch files (including the autoexec.bat file).

### **Setting up the environment variables**

An environment variable is a special system symbol that the debugger uses for finding or obtaining certain types of information. The debugger uses three environment variables, D\_DIR, D\_SRC, and D\_OPTIONS. Use the following instructions to set up these environment variables:

- Identify the 370hll directory with D\_DIR. Enter:

```
SET D_DIR=C:\370HLL
```

(Be careful not to precede the equal sign with a space.)

This directory contains auxiliary files (such as init.cmd) that the debugger needs.

- Identify any directories that contain program source files that you'll want to look at while you're debugging code with D\_SRC. Use this format:

```
SET D_SRC=pathname1;pathname2...
```

For example, if your '370 programs were in a directory named *csource* on drive C, the D\_SRC setup would be:

```
SET D_SRC=C:\CSOURCE
```

- Identify the invocation options that you want to use regularly with D\_OPTIONS. Use this format:

**SET D\_OPTIONS=** [*filename*] [*options*]

The *filename* identifies the optional object file for the debugger to load, and the *options* list the options you want to use at invocation. These are the options that you can identify with D\_OPTIONS:

Option	Brief Description	Page
-b	Select a screen size of 80 characters by 43 lines	2-3
-bb	Select a screen size of 80 characters by 50 lines	2-3
-i <i>pathname</i>	Identify additional directories	2-3
-min	Select the minimal debugging mode	2-3
-profile	Enter the profiling environment	2-4
-s	Load the symbol table only	2-4
-t <i>filename</i>	Identify a new initialization file	2-4
-v	Load without the symbol table	2-4

You can override D\_OPTIONS by invoking the debugger with the -x option.

For more information about options, see Section 2.1, *Invoking the Debugger*, page 2-2.

### **Invoking the new or modified batch file**

- If you modify the autoexec.bat file, be sure to invoke it before invoking the debugger for the first time. To invoke this file, enter:

**AUTOEXEC** 

- If you create your own batch file, you must invoke it *before* entering Windows. You'll need to invoke your batch file any time that you power up or reboot your PC. For the purpose of this discussion, assume that this sample batch file is named *initdb.bat*. To invoke this file, enter:

**INITDB** 



## 1.4 Step 3: Verifying the Installation

To ensure that you have correctly installed the debugger software, invoke the debugger and load the sample program:

If you set up an icon for the debugger, follow these steps:

- 1) Start Windows.
- 2) Open the program group that contains the debugger icon.
- 3) Double-click on the debugger icon.
- 4) When the debugger window appears, enter the following from the command line:

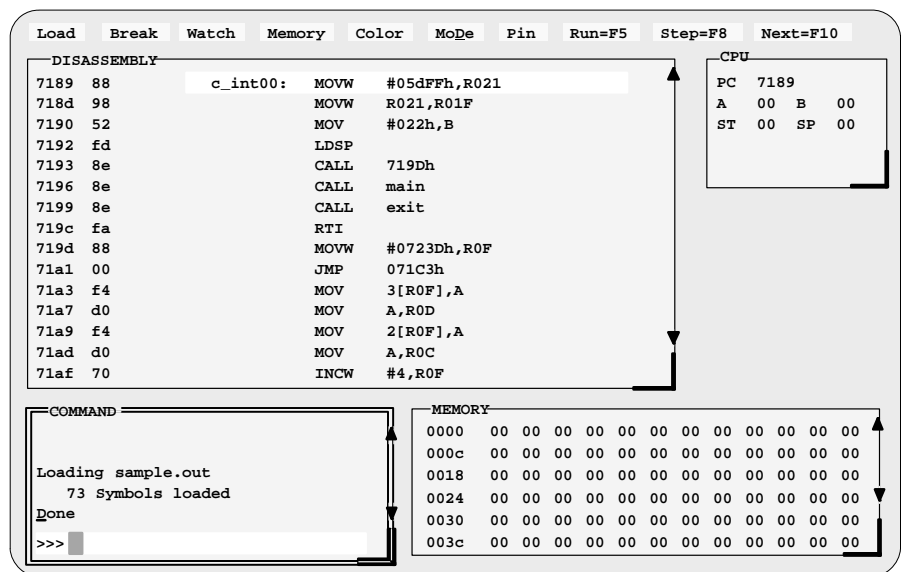
```
load sample
```

If you did not set up an icon for the debugger, follow these steps:

- 1) Start Windows.
- 2) In the Program Manager or File Manager, select Run... from the File menu.
- 3) In the Command Line field of the Run dialog box, enter:

```
c:\370h11\sim370w sample
```

You should see a display similar to this one:



### *Step 3: Verifying the Installation*

---

If you see a similar display, you have correctly installed your debugger.

If you don't see a display, your debugger may not be installed properly. Go back through the installation instructions and be sure that you have followed each step correctly; then reenter the command above.

---

**Notes:**

- 1) Using Windows, you can freely move or resize the debugger display on the screen. If the resized display is bigger than the debugger requires, the extra space is not used. If the resized display is smaller than required, the display is clipped. Note that when the display is clipped, it can't be scrolled.
  - 2) You should run Windows in either the standard mode or the 386 enhanced mode to get the best results.
-

# Simulator Features

This chapter tells you how to invoke the simulator version of the debugger and introduces the simulator's basic features. When you finish reading this chapter, refer to the *TMS370 Family C Source Debugger User's Guide*. The *TMS370 Family C Source Debugger User's Guide* discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. The book also includes a tutorial that introduces basic features of the debugger.

Topic	Page
2.1 Invoking the Debugger .....	2-2
2.2 A Sample Memory Map for the Simulator .....	2-5
2.3 Identifying Usable Memory Ranges .....	2-6
2.4 Simulating I/O Space .....	2-9
2.5 Simulating Interrupts .....	2-11
2.6 Using Predefined Constants With Conditional Commands .....	2-14
2.7 Benchmarking .....	2-15
2.8 Profiling Code Execution .....	2-15
2.9 Debugger Messages .....	2-16

## 2.1 Invoking the Debugger

Here's the basic format for the command that invokes the debugger:

<b>sim370w</b> [ <i>filename</i> ]    [ <i>options</i> ]
--

**sim370w**      is the command that invokes the debugger.

*filename*      is an optional parameter naming an object file that the debugger loads into memory during invocation. The debugger looks for the file in the current directory; if the file isn't in the current directory, you must supply the entire pathname. If you don't supply an extension for the filename, the debugger assumes that the extension is .out.

*options*        supply the debugger with additional information. Table 2–1 summarizes the available options.

You can use the D\_OPTION environment variable to specify files and options that you use often. (See the *Setting up the environment variables* subsection starting on page 1-7.) You can also specify options that you use often on your debugger command line. (See the *Using a program-item icon* subsection on page 1-5 for information about modifying your command line.)

Table 2–1. Summary of Debugger Options

Option	Brief Description	Page
–b	Select a screen size of 80 characters by 43 lines	2-3
–bb	Select a screen size of 80 characters by 50 lines	2-3
–i <i>pathname</i>	Identify additional directories	2-3
–min	Select the minimal debugging mode	2-3
–profile	Enter the profiling environment	2-4
–s	Load the symbol table only	2-4
–t <i>filename</i>	Identify a new initialization file	2-4
–v	Load without the symbol table	2-4
–x	Ignore D_OPTIONS	2-4

### Selecting the screen size (**-b** and **-bb** options)

By default, the debugger uses an 80-character-by-25-line screen. You can use one of the options in Table 2–2 to specify a different screen size.

Table 2–2. Screen Size Options

Option	Description	Display
-b	80 characters by 43 lines	Any EGA or VGA display
-bb	80 characters by 50 lines	VGA only

The **-b** and **-bb** options override the screen size specified in the *init.clr* file.

### Identifying additional directories (**-i** option)

The **-i** option identifies additional directories that contain your source files. Replace *pathname* with an appropriate directory name. You can specify several pathnames; use the **-i** option as many times as necessary. For example:

```
sim370w -i path1 -i path2 -i path3 . . .
```

Using **-i** is similar to using the `D_SRC` environment variable (see the *Setting up the environment variables* subsection starting on page 1-7). If you name directories with both **-i** and `D_SRC`, the debugger first searches through directories named with **-i**. The debugger can track a cumulative total of 20 paths (including paths specified with **-i**, `D_SRC`, and the debugger `USE` command).

### Selecting the minimal debugging mode (**-min** option)

The debugger automatically displays whatever code is currently running: assembly language or C. Depending on the code that is currently running, the debugger displays various windows, such as the `DISASSEMBLY`, `COMMAND`, `CPU`, `MEMORY`, or `CALLS` window.

The debugger has a *minimal* debugging mode that displays the `COMMAND`, `WATCH`, and `DISP` windows only. The `WATCH` and `DISP` windows are displayed only if you cause them to display (by entering the `WA` or `DISP` commands). Minimal mode may be useful when you need to debug a memory problem.

To invoke the debugger and enter minimal mode, use the **-min** option.

For more information about debugger windows, see the *TMS370 Family C Source Debugger User's Guide*.

### **Entering the profiling environment (*-profile* option)**

The *-profile* option allows you to bring up the debugger in a profiling environment so that you can collect statistics about code execution. Only a subset of the basic debugger features is available in the profiling environment.

For more information about using the profiler, see the *TMS370 Family C Source Debugger User's Guide*.

### **Loading the symbol table only (*-s* option)**

If you supply a *filename* when you invoke the debugger, you can use the *-s* option to tell the debugger to load only the file's symbol table (without the file's object code). This option is useful in a debugging environment in which the debugger cannot, or need not, load the object code (for example, if the code is in ROM). Using this option is similar to loading a file by using the debugger's SLOAD command (described in the *TMS370 Family C Source Debugger User's Guide*).

### **Identifying a new initialization file (*-t* option)**

The *-t* option allows you to specify an initialization command file to use instead of *init.cmd*. The format for the *-t* option is:

```
-t filename
```

### **Loading without the symbol table (*-v* option)**

The *-v* option prevents the debugger from loading the entire symbol table when you load an object file. The debugger loads only the global symbols and later loads local symbols as it needs them. This speeds up the loading time and consumes less memory.

The *-v* option affects all loads, including those performed when you invoke the debugger and those performed with the LOAD command within the debugger environment.

### **Ignoring *D\_OPTIONS* (*-x* option)**

The *-x* option tells the debugger to ignore any information supplied with *D\_OPTIONS*. For more information about *D\_OPTIONS*, refer to the *Setting up the environment variables* subsection, starting on page 1-7.

## 2.2 A Sample Memory Map for the Simulator

Because you must define a memory map before you can run any programs, it's convenient to define the memory map in an initialization batch file. Figure 2–1 (a) shows a sample of memory map commands that you could include in an initialization batch file for the simulator.

The MA commands (described on page 2-6) define valid memory ranges and identify the read/write characteristics of the memory ranges. The MAP command enables mapping. (Note that by default, mapping is enabled when you invoke the debugger.) Figure 2–1 (b) illustrates the memory map defined by the MA commands in Figure 2–1 (a).

For more information about memory mapping and the initialization batch file, see the *TMS370 Family C Source Debugger User's Guide*.

Figure 2–1. Sample Memory Map for Use With a '370 Simulator

(a) Memory map commands

```

MA 0x0,0x100,iram
MA 0x100,0x100,xram
MA 0x1010,0x10,iram
MA 0x1020,0x10,iram
MA 0x1030,0x10,siper
MA 0x1040,0x10,tiper
MA 0x1050,0x10,siper
MA 0x1060,0x10,tiper
MA 0x1070,0x10,iram
MA 0x2000,0x1000,xram
MA 0x4000,0x4000,xrom
    
```

(b) Memory map for '370 local memory

0x0000 to 0x00FF	Internal RAM
0x0100 to 0x01FF	External RAM
0x0200 to 0x100F	Reserved
0x1010 to 0x101F	Internal RAM
0x1020 to 0x102F	Internal RAM
0x1030 to 0x103F	Serial internal peripheral frame
0x1040 to 0x104F	Timer internal peripheral frame
0x1050 to 0x105F	Serial internal peripheral frame
0x1060 to 0x106F	Timer internal peripheral frame
0x1070 to 0x107F	Internal RAM
0x1080 to 0x1FFF	Reserved
0x2000 to 0x2FFF	External RAM
0x3000 to 0x3FFF	Reserved
0x4000 to 0x7FFF	External ROM

## 2.3 Identifying Usable Memory Ranges

The debugger's MA (memory map add) command identifies valid ranges of target memory. With the MA command, you can define two basic kinds of memory:

- Internal memory* simulates access to memory locations that are internal to the simulated TMS370 device.
- External memory* simulates access to a memory expansion system.

The syntax of the MA command is:

**ma** *address, length, type*

- The *address* parameter defines the starting address of a range. This parameter can be an absolute address, any C expression, the name of a C function, or an assembly language label.

A new memory map must not overlap an existing entry. If you define a range that overlaps an existing range, the debugger ignores the new range and displays this error message in the display area of the COMMAND window:

```
Conflicting map range
```

- The *length* parameter defines the length of the range in bytes. This parameter can be any C expression.
- The *type* parameter identifies the read/write characteristics and locations of the memory range. The *type* must be one of the keywords shown in Table 2–3.

Table 2–3. Keywords for Use With the Type Parameter

To identify this kind of memory	Use this keyword as the <i>type</i> parameter
Read-only memory	<b>R, ROM, EROM</b>
Read-only external (expansion) memory	<b>XROM</b>
Read-only internal memory	<b>IROM</b>
Read/write memory	<b>RW, RAM, ERAM</b>
Read/write external memory	<b>XRAM</b>
Read/write internal memory	<b>IRAM</b>

**Note:** You cannot use the simulator to simulate EPROM/EEPROM memory or memory that uses wait states.



Table 2–3. Keywords for Use With the Type Parameter (Continued)

To identify this kind of memory	Use this keyword as the <i>type</i> parameter
Read/write serial peripheral frame in memory	<b>SERW, SEPER</b>
Read/write serial peripheral frame in internal memory	<b>SIRW, SIPER</b>
Read/write timer peripheral frame in memory	<b>TERW, TEPER</b>
Read/write timer peripheral frame in internal memory	<b>TIRW, TIPER</b>
Inaccessible memory	<b>PROTECT</b>

**Note:** You cannot use the simulator to simulate EPROM/EEPROM memory or memory that uses wait states.

Be sure that the map ranges that you specify in a common object file format (COFF) file match those that you define with the MA command. Moreover, a command sequence such as:

```
ma x,y,ram; ma x+y,z,ram
```

does not equal

```
ma x,y+z,ram
```

If you were planning to load two COFF blocks, where the first block spanned the length of *y* and the second block spanned the length of *z*, you would use the first MA command example. However, if you were planning to load a COFF block that spanned the length of *y + z*, you would use the second MA command example.

Alternatively, you could turn memory mapping off during a load by using the MAP OFF command. Although the MAP OFF command can be useful, you need to be sure that you use it correctly. See the *Defining a Memory Map* chapter of the *TMS370 Family C Source Debugger User's Guide* for more information about using the MAP OFF command.

### **Restrictions on usable memory ranges**

The following restrictions apply to identifying usable memory ranges:

- Both the starting address and the length of a memory range that you define with the MA command must be a multiple of 16 bytes. If you define a range that is not a multiple of 16 bytes, the debugger ignores the new range and displays this error message in the display area of the COMMAND window:  
`Illegal mapping granularity`  
If you're defining a peripheral frame, this restriction does not apply.
- You cannot specify more than 20 memory ranges.
- Memory locations within the address range 0x1000 to 0x10FF are dedicated to peripheral frames. This range can be simulated, except for addresses 0x1000 and 0x1010, which are used by the simulated device.

## 2.4 Simulating I/O Space

The '370 simulator allows you to simulate peripheral I/O port accesses. To do so, use the MC command to connect a peripheral port to an input or output file. This simulates reads and writes of peripheral data by allowing you to read data in from a file and/or write data out to a file.

### **Connecting a peripheral I/O port**

The MC (memory connect) command connects a peripheral port address to an input or output file. The syntax for this command is:

```
mc portaddress, length, filename, {READ | WRITE}
```

- The *portaddress* parameter defines the address of the peripheral port. This parameter can be an absolute address, any C expression, the name of a C function, or an assembly language label.

The *portaddress* must be previously defined with the MA command as the beginning address of a range of memory, and that range must have a type of **SEPER**, **SIPER**, **SERW**, or **SIRW**. The address range defined for the peripheral frame must be between 0x1000 to 0x10FF.

- The *length* parameter is the length (in bytes) of the file that you are connecting to the peripheral port address.
- The *filename* parameter can be any filename. If you connect a port to read from a file, the file must exist, or the MC command will fail.
- The final required parameter is specified as **READ** or **WRITE** and defines how the file will be used (for input or output, respectively).

The file is accessed (in read or write mode) during the execution of any instruction that reads from or writes to the address of the associated memory-mapped peripheral frame.

Example 2–1 shows how an input peripheral port can be connected to an input file named in.dat.

*Example 2–1. Connecting an Input Peripheral Port to an Input File*

Assume that the file `in.dat` contains words of data in hexadecimal format, one per line, like this:

```
0A
10
20
.
.
.
```

Notice that each line starts with a two-digit hex value; anything other than that on the line is ignored and assumed to be a comment.

These two debugger commands set up and connect a peripheral port:

```
MA    0x1020,0x10,SEPER           Configure address 0x1020
                                     as a read-only peripheral frame
MC    0x1020,1,in.dat,READ        Open file in.dat and
                                     connect to port address 0x1020
```

Assume that this '370 instruction is part of your '370 program. The instruction reads from the `in.dat` file:

```
MOV    P022,A                    MOV instruction reads from file
                                     and loads the read data in register A
```

**Disconnecting a peripheral I/O port**

Before you can use the MD command to delete a peripheral frame from the memory map, you must use the MI command to disconnect the peripheral port. The MI (memory disconnect) command disconnects a file from a peripheral I/O port. The syntax for this command is:

**mi** *portaddress*, {**READ** | **WRITE**}

- The *portaddress* identifies the port that will be closed.
- The second parameter, **READ** or **WRITE**, must match the parameter that was used when the port was connected with the MC command.

## 2.5 Simulating Interrupts

The '370 simulator allows you to simulate and monitor internal and external interrupt signals and to specify at what clock cycle you want an interrupt to occur. To do this, you create a data file and connect it to one of two interrupt levels (or *pins*), LEV1 or LEV2.

### Setting up your input file

In order to simulate interrupts, you must first set up an input file that lists interrupt intervals. Your file must contain a clock cycle and an offset value in the following format:

```
[clock cycle, offset] [ ( [clock cycle, offset]...) ] [ rpt {n | EOS} ]
```

Note that the parentheses are optional and are used for grouping interrupt values, while repeating a particular pattern.

- The *clock cycle* parameter represents the CPU clock cycle where you want an interrupt to occur. Note that the square brackets around *clock cycle* and *offset* are part of the syntax and must be included.

You can have two types of CPU clock cycles:

- **Absolute.** To use an absolute clock cycle, your cycle value must represent the actual CPU clock cycle where you want to simulate an interrupt. For example:

```
[12,0xfa] [34,0xfc] [56,0xfa] [78,0xfc]
```

An interrupt signal is simulated at the 12th, 34th, 56th, and 78th CPU clock cycles. Notice that no operation is done to the clock cycle value; the interrupt occurs exactly as the clock cycle value is written.

- **Relative.** You can also select a clock cycle that is relative to the time at which the last event occurred. For example:

```
[12,0xfa] [+34,0xfc] [55,0xfa] [+20,0xfc]
```

An interrupt signal is simulated at the 12th, 46th (12+34), 55th, and 75th (55+20) CPU clock cycles. A plus sign (+) before a clock cycle adds that value to the clock cycle preceding it. As shown in this example, you can mix both relative and absolute cycle values in your input file.

You can choose to use a relative clock cycle value as your first clock cycle parameter. In this case, the clock cycle value is relative to 0.

- The *offset* parameter, when added to the interrupt table base address, represents the interrupt vector location. The debugger reads this interrupt vector and loads it into the PC. The offset value must be an 8-bit even number.

For example, the debugger reads your input file and is instructed to generate an interrupt on  $\overline{\text{LEV1}}$ . Assume that the base address in the interrupt table is 0x7F00 and that your offset value is 0x00FA. The debugger reads the interrupt vector table at locations 0x7FFA and 0x7FFB to retrieve the starting value of the PC.

- The **rpt** {*n* | **EOS**} parameter is optional and represents a repetition value. You can have two forms of repetition to simulate interrupts:

- Repeat a fixed number of times.** You can format your input file to repeat a particular pattern for a fixed number of times. For example:

```
[20,0x3e] ([+5,0x24] [+10,0x10]) rpt 4
```

The values inside of the parentheses represent the portion that is repeated. Therefore, an interrupt signal is simulated at the 20th, 25th (20+5), 35th (25+10), 40th (35+5), 50th (40+10), 55th (50+5), 65th (55+10), 70th (65+5), and 80th (70+10) clock cycles.

Make sure that *n* is a positive integer value.

- Repeat to the end of simulation.** To repeat the same pattern throughout the simulation, add the string EOS to the line. For example:

```
([+100,0xf0]) rpt EOS
```

An interrupt is generated every 100 clock cycles.

### Programming the simulator

After you have created your input file, you can use debugger commands to:

- Connect the interrupt pin to your input file
- List the interrupt pins
- Disconnect the interrupt pin from your input file

Use these commands as described below, or use them from the PIN pulldown menu at the top of the debugger display.

To attach your input file to the interrupt pin, use the PINC command:

**pinc** *pinname*, *filename*

- The *pinname* parameter identifies the interrupt pin and must be either LEV1 or LEV2.
- The *filename* parameter is the name of your input file. Make sure you have set up your input file as described in the previous subsection.

For example, to connect the input file myfile to the LEV1 interrupt pin, you would enter:

```
pinc lev1, myfile
```

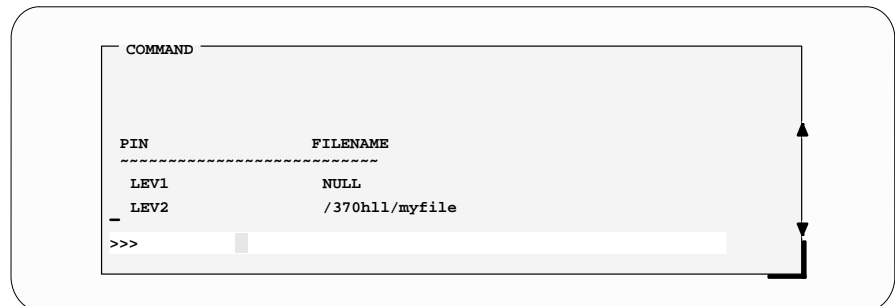
**Note: Do Not Connect a Pin to an Empty File**

Do not try to connect a pin to an empty file. Make sure you set up the file as described in the *Setting up your input file* subsection, starting on page 2-11.

To verify that your input file is connected to the correct pin, use the PINL command. The syntax for this command is:

**pinl**

The PINL command displays all of the unconnected pins first, followed by the connected pins. For a connected pin, the simulator displays the name of the pin and the absolute pathname of the file in the COMMAND window.



When you want to connect another file to an interrupt pin, the PINL command is useful for looking up an unconnected pin.

To end the interrupt simulation, you must disconnect the pin. You can do this with the PIND command:

**pind** *pinname*

The *pinname* parameter identifies the interrupt pin and must be either LEV1 or LEV2. The PIND command detaches the file from the interrupt pin. After executing this command, you can connect another file to the same pin.

## 2.6 Using Predefined Constants With Conditional Commands

In batch files, you can control the flow of debugger commands by choosing to conditionally execute debugger commands or set up a looping situation by using the IF/ELSE/ENDIF or LOOP/ENDLOOP commands, respectively. The *TMS370 Family C Source Debugger User's Guide* describes these commands.

When you use the IF/ELSE/ENDIF command sequence, you can use some predefined constants. These constants evaluate to 0 (false) or 1 (true). Table 2–4 shows the constants and their corresponding tools.

Table 2–4. Predefined Constants for Use With Conditional Commands

Constant	Debugging Tool
\$\$XDS22\$\$	Emulator
\$\$SIM\$\$	Simulator
\$\$ABD\$\$	Application board
\$\$CDT370\$\$	Compact development tool

One way you can use these predefined constants is to create an initialization batch file that works for any debugger tool. This is useful if you are using, for example, both an XDS/22™ emulator and the simulator. To do this, you can set up the following batch file:

```
if $$XDS22$$
echo Invoking initialization batch file for XDS.
use \370tools
take init.cmd
.
.
endif

if $$SIM$$
echo Invoking initialization batch file for simulator.
use \370hll
take init.cmd
.
.
endif
.
.
```

In this example, the debugger executes only the initialization commands that apply to the debugging tool that you invoke.



## 2.7 Benchmarking


The simulator version of the debugger allows you to keep track of the number of CPU clock cycles consumed by a particular section of code. The debugger maintains the count in a pseudoregister named *CLK*. This process is referred to as *benchmarking*.

Benchmarking code is a multiple-step process:

**Step 1:** Set the program counter (PC) value at the statement that marks the beginning of the section of code that you'd like to benchmark. (You can do this either by editing the PC value at the command line or by setting a software breakpoint at the statement you'd like to benchmark.)

**Step 2:** Set a software breakpoint at the statement that marks the end of the section of code you'd like to benchmark.

**Step 3:** Now enter the RUNB command:

```
runb 
```

When the processor halts at the second breakpoint, the value of CLK is valid. To display it, use the ? command or enter it into the WATCH window with the WA command. This value is valid until you enter another RUN command.

### Notes:

- 1) The value in CLK is valid only after using a RUNB command that is terminated by a software breakpoint. (The maximum value for CLK is 65535.)
- 2) When programming in C, do not use a variable named CLK.

For more information about benchmarking and using software breakpoints, see the *TMS370 Family C Source Debugger User's Guide*.

## 2.8 Profiling Code Execution

The simulator version of the debugger includes a second debugger environment: a profiling environment. The profiling environment provides a method for collecting execution statistics about specific areas in your code. This gives you immediate feedback on your application's performance.

The profiling environment is described in the *TMS370 Family C Source Debugger User's Guide*.

## 2.9 Debugger Messages

Appendix C, *Debugger Messages*, in the *TMS370 Family C Source Debugger User's Guide* contains an alphabetical listing of the progress and error messages that the debugger might display in the display area of the COMMAND window. In addition to the messages listed in that appendix, you may encounter the following messages.

### A

#### Area restricted to peripherals

*Description* You attempted to add a nonperipheral frame range within the dedicated peripheral area (0x1000 to 0x10FF).

*Action* Reenter the MA command, specifying a range that is *not* within 0x1000 to 0x10FF.

### C

#### Cannot connect pin

*Description* You attempted to connect a pin to a file that the debugger cannot open.

*Action* Be sure that the filename was typed correctly. If it was, then:

- 1) Check the access rights of the file and/or the directory that contains the file.
- 2) Reenter the command and specify full path information with the filename.

#### Cannot disconnect pin

*Description* You tried to disconnect the input file from a pin that was not previously connected to that pin.

*Action* Use the PINL command to list all of the pins and the files connected to them. Use the PIND command to reenter the correct pin name and filename.

#### Cannot map external peripheral into area

*Description* You attempted to add an external peripheral frame range that is outside of the dedicated external peripheral area (0x1000 to 0x10FF).

*Action* Reenter the MA command, specifying a range that is within 0x1000 to 0x10FF.

**Cannot map internal peripheral into area**

*Description* You attempted to add an internal peripheral frame range that is outside of the dedicated internal peripheral area (0x1000 to 0x10FF).

*Action* Reenter the MA command, specifying a range that is within 0x1000 to 0x10FF.

**Cannot map memory into peripheral area**

*Description* One of the following occurred:

- The debugger tried to access the memory range dedicated to peripheral frames (0x1000 to 0x10FF).
- You attempted to define emulator, internal, external, or read-only memory in the memory range dedicated to peripheral frames (0x1000 to 0x10FF).

*Action* Remap the reserved memory accesses.

**Cannot map port address**

*Description* You attempted to do a connect/disconnect on an illegal port address.

*Action* Verify that the address you specified is a valid port address.

**Cannot open port file**

*Description* You attempted to connect a port to a file that the debugger cannot open.

*Action* Be sure that the filename was typed correctly. If it was, then:

- 1) Check the access rights of the file and/or the directory that contains the file.
- 2) Reenter the command and specify full path information with the filename.

**F****File already tied to port**

*Description* You attempted to connect to an address that already has a file connected to it.

*Action* Connect the file to a mapped port that is not already connected to another file.

### **File already tied to this pin**

*Description* You attempted to connect an input file to an interrupt pin that already has a file connected to it.

*Action* Use the PINC command to connect the file to another interrupt pin that is not connected to a file.

### **File does not exist**

*Description* The port file could not be opened for reading.

*Action* Be sure that the file exists as named. If it does, enter the USE command to identify the file's directory.

### **Files must be disconnected from ports**

*Description* You attempted to delete a memory map that has files connected to it.

*Action* You must disconnect a port with the MI command before you can delete it from the memory map.



### **Illegal mapping granularity**

*Description* You attempted to do one of the following:

- Add a memory range that does not start on an address that is a multiple of 16 bytes or does not have a length that is a multiple of 16 bytes. This restriction does not apply when you're defining a peripheral range or an expansion memory range
- Add an expansion memory range that does not start on an address that is a multiple of 4K bytes or does not have a length that is a multiple of 4K bytes

*Action* Choose the appropriate action for the type of memory range that you are trying to add:

- For a nonperipheral and nonexpansion memory range, reenter the MA command and specify a range that starts on an address that is a multiple of 16 bytes and has a length that is a multiple of 16 bytes.
- For an expansion memory range, reenter the MA command and specify a range that starts on an address that is a multiple of 4K bytes and has a length that is a multiple of 4K bytes.

**Illegal memory access**

- Description* The CPU attempted a read access to an inaccessible or undefined memory range or a write access to an inaccessible, undefined, or read-only memory range.
- Action* Check your memory map to be sure that you access valid memory.

**Illegal syntax for this type of pin**

- Description* You attempted to connect a pin to a file that doesn't have a valid syntax.
- Action* Correct the input file. Refer to Section 2.5, *Simulating Interrupts*, for more information about creating an input file. After you've modified the file, use the PINC command to reconnect the file.

**Illegal write access**

- Description* Your program attempted to write to an unmapped address or to an address that does not have write permission.
- Action* Check your memory map and modify it as necessary.

**Input number too big at line number**

- Description* You attempted to connect a pin to a file that has an input value greater than 0x00FF FFFF. The input value can be 0 through 0x00FF FFFF only.
- Action* Correct the input file and use the PINC command to reconnect the file.

**Interrupt vector offsets must be even**

- Description* In the file that you connected to a pin, there is an incorrect offset value (an odd value) associated with a clock cycle. This error is detected during program execution.
- Action* Disconnect the file that contains the error, modify the file to correct the offset value, then use the PINC command to reconnect the file. Next, enter the RESET command to reset the value of the CLK pseudoregister, enter the RESTART command, and rerun your program.

### Invalid memory attribute

*Description* The third parameter of the MA command specifies the type, or attribute, of the block of memory that MA adds to the memory map. The parameter entered did not match one of the valid attributes.

*Action* Reenter the MA command. Use one of the following valid parameters to identify the memory type:

Parameter	Type of Memory
R, ROM, EROM	Read-only memory
XROM	Read-only external (expansion) memory
IROM	Read-only internal memory
RW, RAM, ERAM	Read/write memory
XRAM	Read/write external memory
IRAM	Read/write internal memory
SERW, SEPER	Read/write serial peripheral frame
SIRW, SIPER	Read/write serial peripheral frame
TERW, TEPER	Read/write timer peripheral frame
TIRW, TIPER	Read/write timer memory
PROTECT	Inaccessible memory

### Invalid memory attributes combination

*Description* While attempting to add a memory range, you used an invalid memory-type keyword.

*Action* Refer to Section 2.3, *Identifying Usable Memory Ranges*, on page 2-6 for a complete list of valid memory-type keywords. Reenter the MA command with a valid keyword.

**N****Nesting of pending interrupts cannot exceed 4**

*Description* The debugger cannot handle more than four nested levels of pending interrupts (interrupt requests that occur during the execution of an interrupt routine).

*Action* Ensure that the value of the ST register enables this interrupt pin during program execution. If it does, disconnect the file that contains the error, modify the file, then use the PINC command to reconnect the file. Next, enter the RESET command to reset the value of the CLK pseudoregister, enter the RESTART command, and rerun your program.

**Nesting of repeats cannot exceed 100****Number of pending interrupts cannot exceed 100**

*Description* The debugger cannot handle more than 100 pending interrupts.

*Action* Ensure that the value of the ST register enables this interrupt pin during program execution. If it does, disconnect the file that contains the error, modify the file, then use the PINC command to reconnect the file. Next, enter the RESET command to reset the value of the CLK pseudoregister, enter the RESTART command, and rerun your program.

**No file tied to this pin**

*Description* You tried to disconnect the input file from a pin that was not previously connected to that pin.

*Action* Use the PINL command to list all of the pins and the files tied to them. Use the PIND command to reenter the correct pin-name and filename.

**P****Pinname not valid for this chip**

*Description* You attempted to connect or disconnect an input file to an invalid interrupt pin.

*Action* Reconnect or disconnect the input file to an unused interrupt pin ( $\overline{\text{LEV1}}$  or  $\overline{\text{LEV2}}$ ).

## R

### Read not allowed for port

*Description* You attempted to connect a file for input operation to an address that is not configured for read.

*Action* Remap the port or correct the access in your source code.

## S

### Syntax error at line number: ?

*Description* You attempted to connect a pin to a file that doesn't have a valid syntax.

*Action* Correct the input file. Refer to Section 2.5, *Simulating Interrupts*, on page 2-11 for more information about creating an input file. After you've modified the file, use the PINC command to reconnect the file.

## W

### Write not allowed for port

*Description* You attempted to connect a file for output operation to an address that is not configured for write.

*Action* Either change the '370 software to write to a port that is configured for write, or change the attributes of the port.