User's Guide

GLS™ Symbolic Debugger (mdb™)

216–856007–001

AEG

MODCOMP

AEG

User's Guide

GLS™ Symbolic Debugger (mdb™)

MODCOMP

Manual History

Manual Order Number: 216–856007–001

Title: User's Guide, GLS Symbolic Debugger (mdb)

Revision Level	Date Issued	Description
000	11/89	Initial Issue.
001	05/90	Change. Corrected pathnames of installed directories to reflect INSTALL script.

Contents subject to change without notice.

MODULAR COMPUTER SYSTEMS, INC.
1650 WEST McNAB ROAD
P.O. BOX 6099
FORT LAUDERDALE, FL 33340-6099

Preface

Audience

This manual is written for experienced system programmers who are using the General Language System (GLS™) compilers. Users should be familiar with AT&T UNIX® System V or REAL/IX™ Operating System terminology.

Subject

This manual tells how to install and use the GLS Symbolic Debugger (**mdb**™). The **mdb** debug windows incorporate the full functionality of AT&T's UNIX **sdb**(1) command options. A section describing **mdb** rules and syntax is also included.

Product Requirements

Use the GLS Symbolic Debugger with programs processed by one of the GLS compilers. **mdb** runs under the REAL/IX Operating System on the CLASSIC® Tri-Dimensional™ Model 97xx computer system.

Related Publications

Refer to the following manuals for additional information. When you order additional manuals, use the manual order number listed below. The most current revision level (REV) will be shipped.

Manual Order Number	Manual Title
215-856001-*REV*	GLS FORTRAN Interface to System Services Library Reference Manual
210-856001-*REV*	GLS FORTRAN Language Reference Manual
216-856005-*REV*	GLS Programming Guide for 97xx Systems
216-856007-*REV*	GLS Symbolic Debugger User's Guide
205-855001-*REV*	REAL/IX Operating System, 97xx Systems Concepts and Characteristics
211-855001-*REV*	REAL/IX Operating System, 97xx Systems Commands and Utilities Reference Manual
211-854002-*REV*	REAL/IX Operating System, 97xx Systems System Calls, Library Routines, and Files Reference Manual

MODCOMP Service and Assistance

MODCOMP® offers a variety of programs and services that demonstrate our commitment to customer satisfaction. Our Technical Education department provides comprehensive hands-on instruction either at our facilities or at customer-designated sites. Our worldwide field service organization is ready to provide installation assistance, free service during the warranty period, and flexible service programs tailored to your requirements.

Questions, Problems, and Suggestions

Your MODCOMP sales and service representatives can help you with any questions, problems, or suggestions you may have regarding our products and services. In addition, for your convenience MODCOMP maintains toll-free telephone numbers at which we can be reached for questions, problems, and suggestions. Please feel free to use the following numbers:

- **For questions, sales information, or suggestions**: in the U.S. and Canada, 1-800-255-2066 (In countries outside the U.S. and Canada, please call your regional sales support office or 1-305-974-1380 extension 1800 worldwide.)

- **For service**: in Florida, 1-800-432-1405; in the U.S., 1-800-327-8928; in Canada, 1-416-890-0666 (In countries outside the U.S. and Canada, please call your regional service/support office.)

- **For Technical Education information**: in the U.S., 1-305-977-1708 (In countries outside the U.S., please call your regional support office.)

For comments about documentation, please use the response form at the back of this manual.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The GLS™ Symbolic Debugger, **mdb**™, uses a special set of symbols[1] to establish a connection between source code and executable instructions. The compile and link process typically destroys the ability to trace back to the source level. **mdb** restores this connection by providing a high level debugging environment with a window interface that allows you to pause execution at breakpoints, single–step through the program, and examine register or memory contents.

mdb is powerful, but easy to access. As few as five commands provide the user with debugging capabilities. Once you are familiar with the basic debugger functions, the **mdb** user interface and command set make it possible to solve program logic problems with minimum effort. Refer to the section "Using mdb" for the **mdb** command descriptions.

The GLS Symbolic Debugger includes the following features:

- A user–friendly window interface
- On–line help facility
- Log file capability
- Command files for commonly used **mdb** command sequences.

[1] Symbols are saved during each stage of the compile, assemble, and link translation process.

The Window–Oriented Screen Interface

Based on the standard UNIX® library **curses**(3X) package, the window interface is terminal independent. This means you can use any intelligent terminal if you have an entry in the **terminfo**(4) directory for that terminal.

When you execute **mdb**, you are presented with the initial *Standard Screen*. This screen includes the *Source, Command Output,* and *Command Input* subwindows.

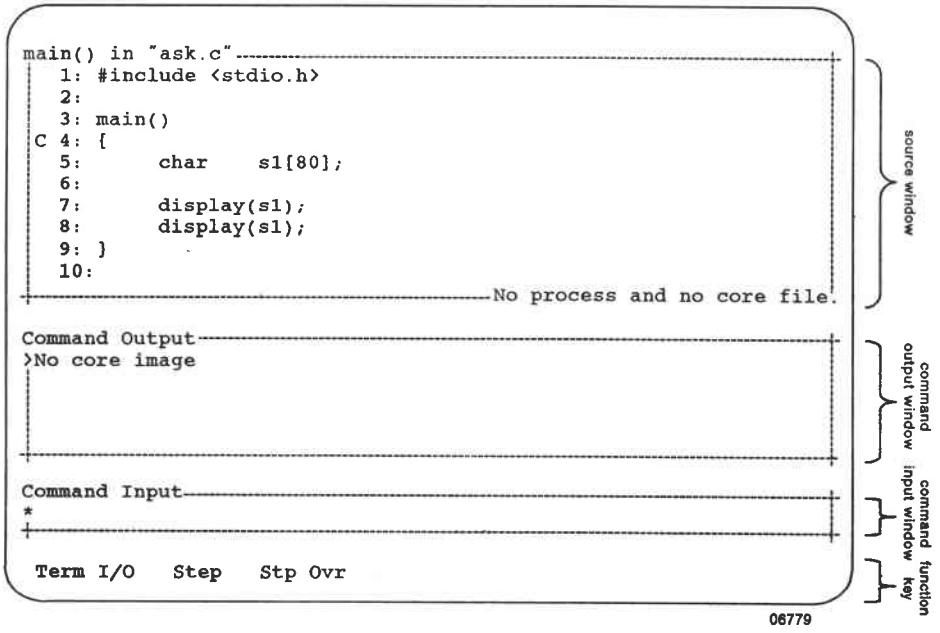

Figure 1–1. Standard Screen

The Source Window displays the current source procedure and file name, active breakpoints in the current procedure, the current line, last line executed, and the top of the stack trace.

The Command Output window displays debug process output. The output window implements a subset of the REAL/IX™ **pg**(1) command options in the scrolling function. Note that the message `>No core image` in the Command Output Window means the user did not request a core image file when executing **mdb**. Refer to the section "Calling mdb" in Chapter 4 .

The Command Input window accepts all **mdb** command line input and supports a history function that allows you to edit and reissue commands.

Window Interface Commands

The GLS Symbolic Debugger package provides full symbolic debugging capabilities, as well as menu-driven screens and window-oriented commands that make **mdb** easy to use.

Control Keys

In this document, a bold character preceded by an uparrow symbol (^) means press the control key while striking the specified character key. The following control keys have been added to provide enhanced functionality:

- ^P dumps the current screen to the file *./sdump.dmb*.
- ^R clears and redraws the screen.
- ^E calls up the on-line help window. Refer to the section "The Window Cycle" for a discussion of the on-line help windows.

Function Keys

Function keys are provided to eliminate keystrokes for the following commands (refer to Figure 1-1). You *must* match the *terminfo* TERM environment variable with your terminal type.

- The `Term I/O` function corresponds to the <F1> key – this opens the Terminal I/O Window.
- The `Step` function corresponds to the <F2> key – this executes the **s** (step) command.
- The `Stp Ovr` function corresponds to the <F3> key – this executes the **S** (stepover) command.

The Window Cycle

mdb supports the ability to cycle from one window to the next; each window includes a pop-up window that provides on-line help.

To access each screen press the escape key (<ESC>). Press ^E to activate the help window for the currently active window. Press the carriage return key (<CR>)[1] to exit a window; this returns you to command input mode. Each time you press <ESC>, you cycle to the next screen.

Cycle Position	Active Window
1	Command Input
2	Command Output
3	System
4	Terminal I/O

[1] The carriage return key is also known as the <ENTER> or <NEWLINE> key.

On-line help is available through the help windows. The help windows provide the user with information and/or instructions that pertain to the currently active window. To access on-line help for each window press ^E. The following screens illustrate the four **mdb** windows and their associated help windows.

- The *history* function allows the user to reissue and edit previous commands to the Command Input Window. Commands are edited using a subset of the **vi**(1) editor.

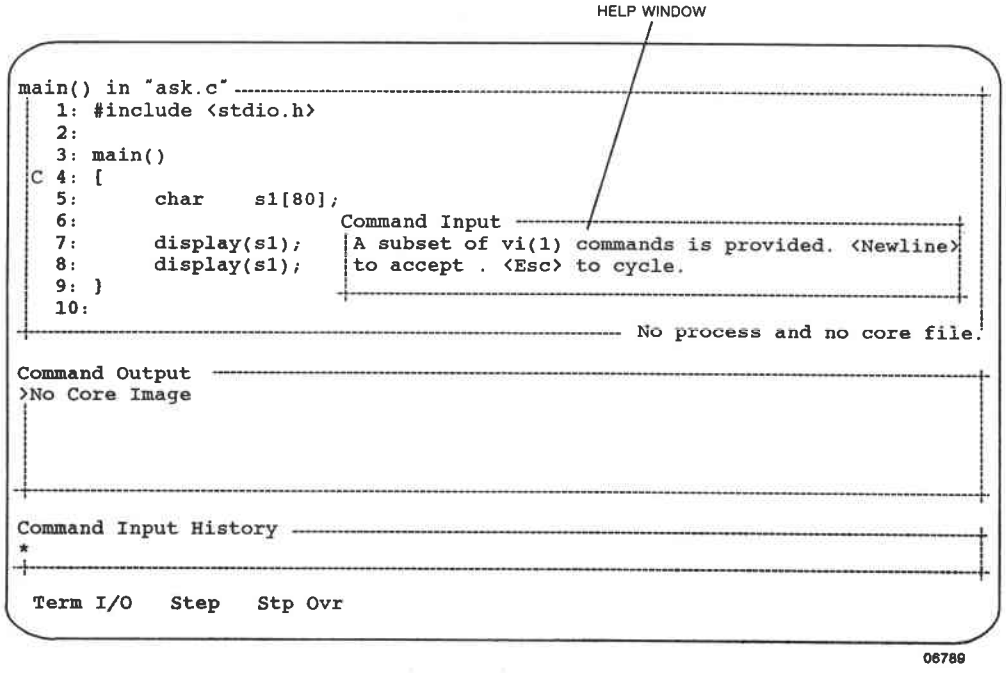

Figure 1-2. Active Command Input Window with History Help Window

Refer to Table 1-1 for the **vi**(1) editor commands that are supported in the Command Input Window.

Table 1–1. Supported vi Commands

vi commands supported by command input	
h, j, k, l	move left, down, up, right,
x	delete a character
ESC	end insert mode
+	move cursor to next line
–	move cursor to previous line
0	move cursor to beginning of line
$	move cursor to end of line
^H	backspace
spacebar	advance one space
a	add after cursor
A	append to end of line
i	Insert before cursor
r*x*	replace character with *x*

- The *scrolling* function allows the user to review debugger output information by moving forward and back in the Command Output window.

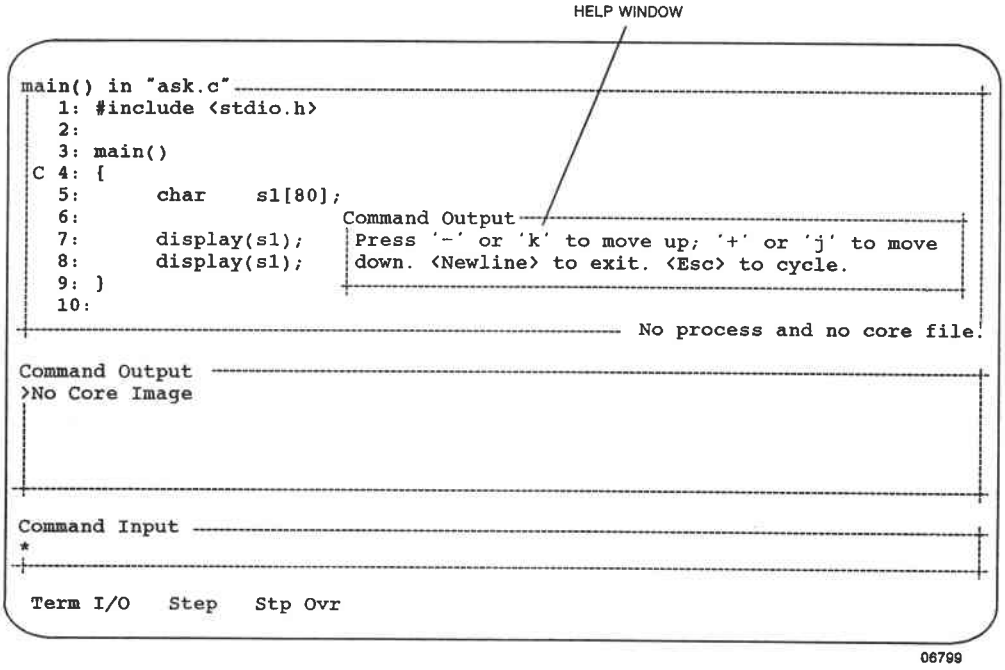

Figure 1–3. Active Command Output Window with Scrolling Function

- The *system default* function accesses the System Window. The System Window allows the user to modify **mdb** system parameters.

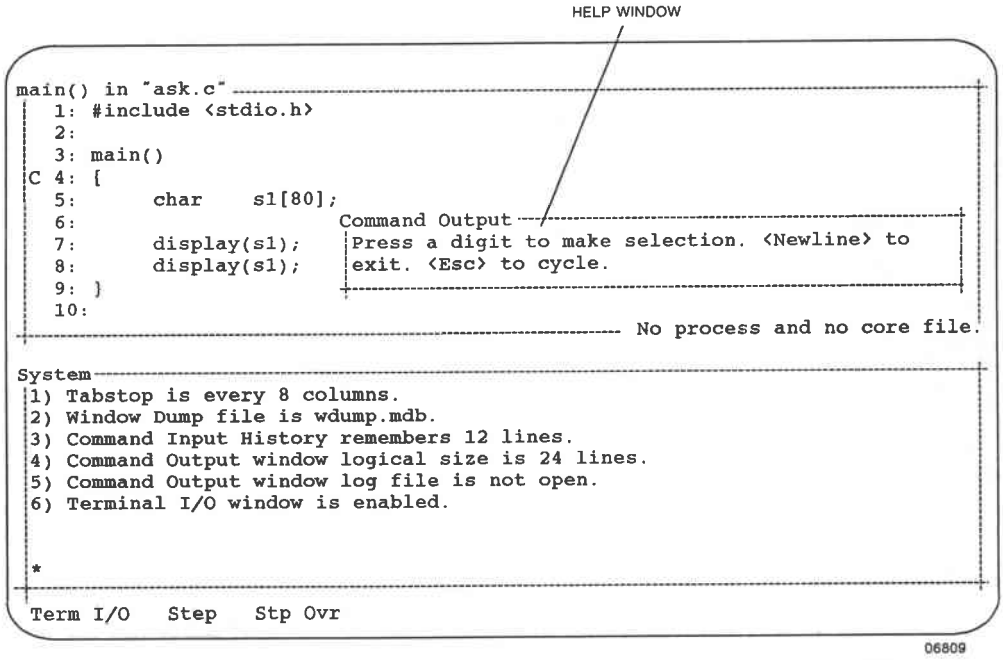

Figure 1–4. Active System Window with Help Window

When you make a selection from the System Window menu, **mdb** provides further instructions and/or waits for the related input. Refer to following sections for additional information about menu items **5** and **6**.

Menu Item 5: Log Files

The contents of the Command Output Window are recorded in the log file. The log file is opened and closed in the System Parameters Window (refer to Figure 1–4). If a log file exists, the name of the log file appears in the lower right–hand corner of the Command Output Window (refer to Figure 1–5).

If you select item number **5** from the System Window menu, you are prompted to enter the pathname. *file*.**log** is like any other REAL/IX™ file. After you create the file, it resides either in the current working directory or in the directory specified by the pathname.

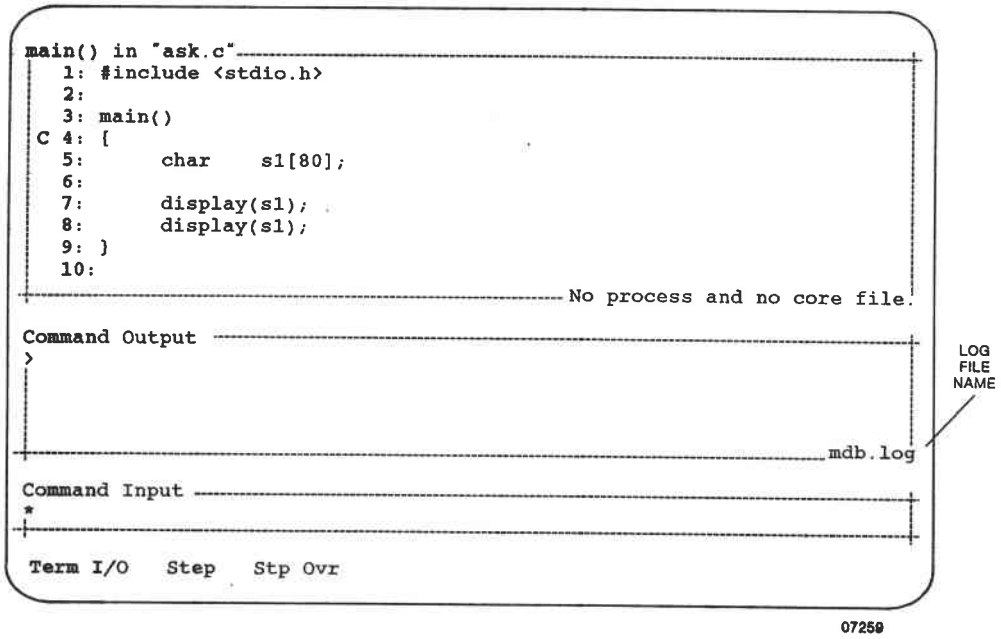

Figure 1–5. Standard Screen with Open Log File

Menu Item 6: Terminal I/O Window

The Terminal I/O Window is toggled to enable or disable in the System Parameters Window (refer to Figure 1–4). When you enable the Terminal I/O Window *stdin*, *stdout*, and *stderr* are piped through the debugger program. When you disable the Terminal I/O Window, your user process has direct access to the terminal driver.

	enabled	disabled
user process I/O saved:	yes	no
control characters converted to readable format:	yes	no
stty(1) terminal setting for <EOF> processed:	no	yes

- The Terminal I/O Window is activated when the user process writes and flushes *stdout* or *stderr* or executes for approximately one second.

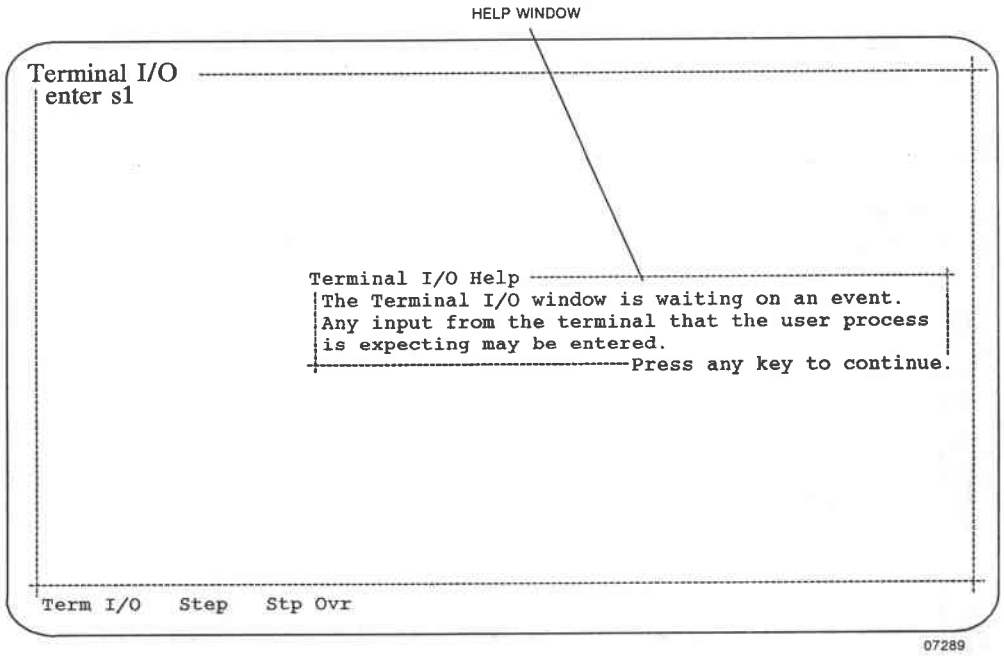

Figure 1–6. Active Terminal I/O Window with Help Window

The Hour Glass Symbol

The **hour** glass symbol indicates that **mdb** is either in process or waiting for an event. When active, the **symbol** appears in the center of the screen (refer to Figure 1–7) and the frames move clockwise once a second, until the process is completed.

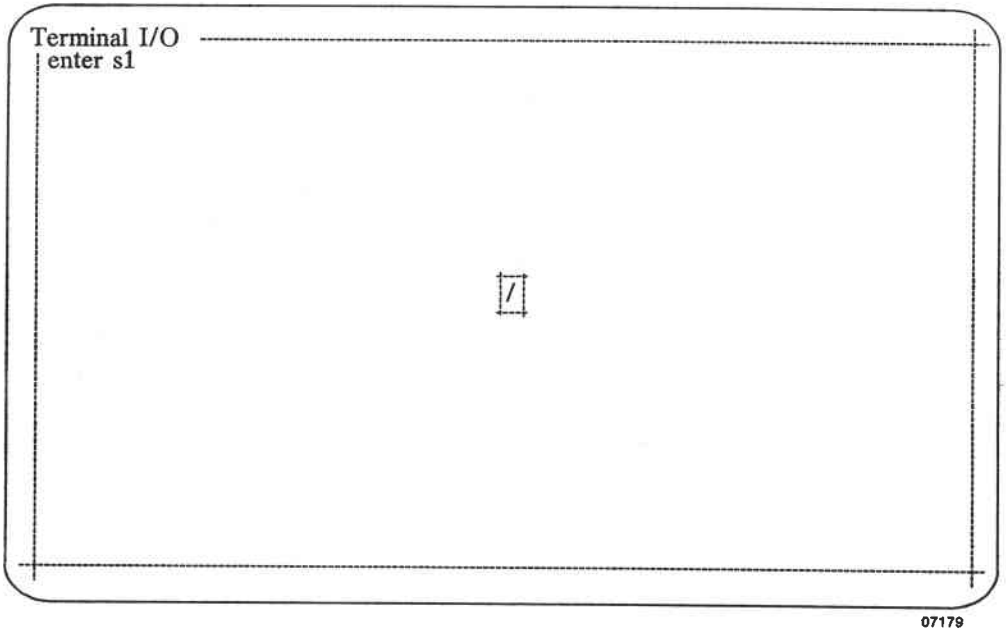

Figure 1–7. An Example of Screen with Hour Glass Symbol

The Error Window

The Error Window automatically displays all system and active window errors. Active window errors indicate an invalid action in the currently active window. Refer to Figure 1–8.

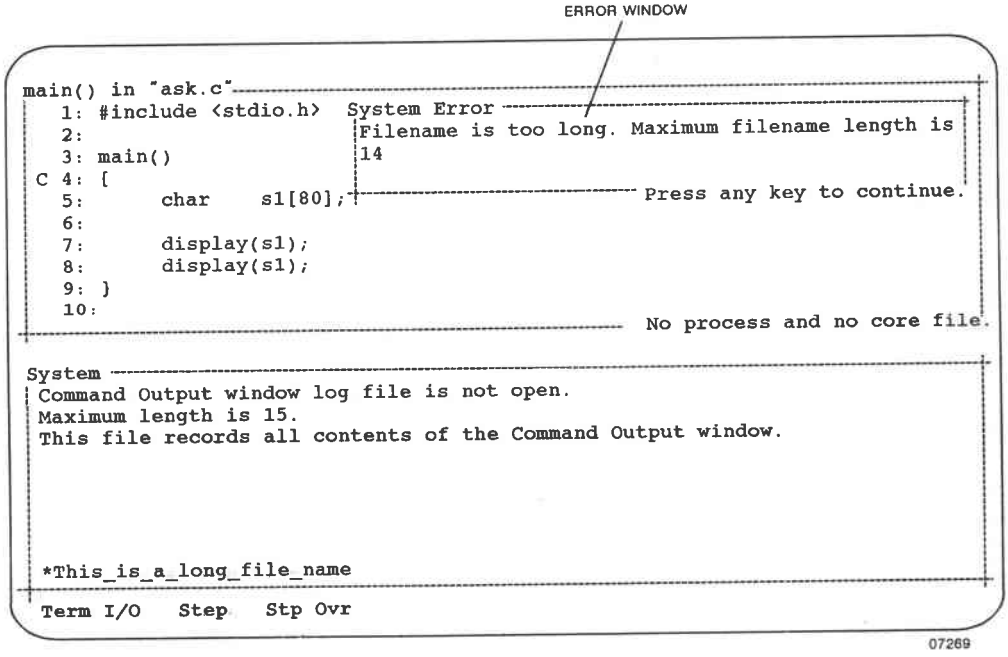

Figure 1–8. Active Error Window

Chapter 2

mdb Features

mdb permits full execution flow control. Breakpoints and single-stepping allow you to observe activity while the program executes. You can create more informative breakpoints and run-time patching by attaching commands to breakpoints.[1] Refer to Chapter 4, "Using mdb", for more information.

Breakpoints

Breakpoints let you examine program status by stopping program execution at specific locations and returning control to the user. When you set breakpoints and execute your program, execution continues until a break is encountered; at that point the word `>Breakpoint` is written to the Command Output Window and program execution is stopped. At this point you can interactively debug the program.

You can associate a command with a breakpoint by entering the command and breakpoint in the Command Input Window as follows:

```
*17b s1;k
```

This breakpoint command is defined as follows:

`17`	go to line 17 in the current source file
`b`	insert a breakpoint command
`s1`	print the contents of the variable *s1*
`;`	separate the first command from the second command on the command line
`k`	stop execution

When you execute this command the contents of variable `s1` appear in the Command Output Window, followed by the word `>Breakpoint`. Refer to Figure 2-1.

[1] To fully utilize **mdb** execution flow control, the source program must be compiled using the **-g** option. The **-g** option causes the compiler to generate additional information about variables and statements in the compiled program. You can use **mdb** without compiling the source code with the **-g** option, but this limits the symbolic debugging capabilities.