MICROTEK

Development Solutions For Embedded System Design

PowerPack® Development System

MICROTEK

Development Systems Division

For x86 Processors

SLD™ Source Level Debugger for the PowerPack® Emulator

User's Manual

MICROTEK INTERNATIONAL
Development Tools
Doc. No. I49-001081
Part No. 15055-000
May 1996

Trademark Acknowledgments

PowerPack is a registered trademark and SLD is a trademark of Microtek International.

IBM, LAN, and OS/2 are trademarks of IBM.

Microsoft is a registered trademark and MS, MS-DOS, and Windows are trademarks of Microsoft Corporation.

NS486SXF is a trademark of National Semiconductor Corporation

Intel is a registered trademark and Intel386 and Intel486 are trademarks of Intel Corporation.

PC-NFS is a registered trademark of Sun Microsystems.

MICROTEK INTERNATIONAL

Development Systems Division
3300 N.W. 211th Terrace
Hillsboro, OR 97124-7136
USA
Tel: (503) 645-7333
Fax: (503) 629-8460
Email: info@microtekintl.com

6, Industry East Road 3
Science-based Industry Park
Hsinchu 30077
Taiwan, ROC
Tel: +886 35 772155
Fax: +886 35 772598
Email: easupport@adara1.adara.com.tw

Web: http://www.microtekintl.com

Contents

Getting Started

The term "PowerPack emulator" refers to any PowerPack® in-circuit emulator for embedded system development. The terms "PP", "SW", and "EA" refer to the PowerPack PP, SW, and EA emulators respectively. The terms "SLD software", "emulator interface", and "debugger software" refer to the SLD™ source-level debugger.

This chapter describes the emulator and debugger documentation, host system requirements, and how to contact Microtek International for information and technical support.

Documentation

The following describes the printed and on-line documentation resources for the PowerPack emulators. The manuals in your emulator package are the *SLD™ Source-Level Debugger User's Manual* (referred to as the *User's Manual*) and either the *PowerPack® EA/SW In-Circuit Emulator Hardware Reference*, the *PowerPack® EA-NS486 In-Circuit Emulator Hardware Reference*, or the *PowerPack® PP In-Circuit Emulator Hardware Reference* (each referred to as the *Hardware Reference* and formerly known as the *Up & Running*). Other, related publications described at the end of this list are not included in your emulator package.

Resource	Chapter	Contents
Hardware Reference	Getting Started	Parts, features, documentation, support
	Software Installation	Configuring your PC or workstation; installing the SLD software
	Hardware Installation	Installing the PowerPack hardware; running the confidence tests
	Tutorial	Practicing basic emulator tasks
	Target Hardware	SAST board schematics; signals
User's Manual	Getting Started	Host sytem requirements; contacting Microtek
How to...	Defining the Debug Environment	Creating a loadfile; starting and exiting the SLD software; configuring memory and registers; using an initialization file
	Debugging in Source	Viewing source code, disassembly, and stack; editing variables; controlling emulation

Debugging in Registers and Memory	Accessing CPU and peripheral signals and numeric or disassembled memory contents
Debugging with Triggers and Trace	Emulation and trace control using triggers; numeric and symbolic address formats
powerpak.ini File	powerpak.ini file contents
Toolbar	Toolbar controls
Shell Window	Shell window contents, controls, commands
Source Window	Source window contents, controls
Variable Window	Variable window contents, controls
Breakpoint Window	Breakpoint window contents, controls
CPU Window	CPU window contents, controls
Stack Window	Stack window contents, controls
Memory Window	Memory window contents, controls
Peripheral Window	Peripheral window contents, controls
Trace Window	Trace window contents, controls
Event Window	Event window contents, controls
Trigger Window	Trigger window contents, controls

Reference

PowerPack SLD Help

For help on using online help, choose How to Use Help from any SLD Help menu or press <F1> twice.

Whether or not the emulator is active, you can invoke the SLD online help from within Windows. Choose the SLD Help icon (shown at left). SLD online help conforms to the standard Windows help interface, as described in your Microsoft Windows documentation.

For help from within the SLD software, choose a Help menu item; or, press <F1> at any time. In most SLD dialog and message boxes, you can choose a Help button for context-sensitive help. In the Shell window, you can list Shell command syntax with a Help command.

Related Publications

Topic	Resource
Windows 3.1; Windows 95; Windows for Workgroups 3.11	Microsoft documentation
Your target processor	Your chip vendor documentation
Your toolchain and loadfile format	Your compiler, assembler, linker, and converter documentation
C++ name mangling	*The Annotated C++ Reference Manual*, Margaret Ellis and Bjarne Stroustrup (Addison-Wesley, 1990)

How to Contact Microtek

To register for technical support and ongoing product information, complete and mail the registration card enclosed with the emulator.

Contact Microtek/DSD to purchase an Extended System Warranty (ESW). An ESW provides firmware, software, and hardware updates and priority service, in addition to repairs.

As a Microtek customer, you can contact Microtek technical support for help with an emulator problem during your warranty period. The email and fax lines are operational 24 hours a day, 7 days a week.

Internet email	csupport@microtekintl.com (technical support) info@microtekintl.com (other information)
World Wide Web	http:\\www.microtekintl.com (product news)
Microtek/DSD, Western USA	(503) 645-7333 voice; (503) 629-8460 fax (voice contact available Monday through Friday, 8:00 am to 5:00 pm USA Pacific Time)
Microtek, Eastern USA	(610) 783-6366 voice; (610) 783-6360 fax (voice contact available Monday through Friday, 8:00 am to 5:00 pm USA Eastern Time)
Microtek, Hsinchu, Taiwan	+886-35-77-2155 voice; +886-35-77-2598 fax (voice contact available Monday through Friday, 8:00 am to 5:00 pm Taiwan Time)
Adara International, Taipei, Taiwan	+886-2-501-6699 voice; +886-2-505-0137 fax (voice contact available Monday through Friday, 8:00 am to 5:00 pm Taiwan Time)

Before you call, please read the *PowerPack® Emulator Problem Report Form* in the SLD on-line help.

When you call, please be at your computer with the SLD software running and have the emulator documentation and filled-out problem report form (printable from the on-line help) nearby.

Host System Requirements and Recommendations

- An Intel486 or Pentium processor based or 100% compatible PC
- Windows 95; or, MS-DOS 5.0 or 6.x with Windows 3.1 or Windows for Workgroups 3.11 running in 386-enhanced mode
- At least 8M bytes of RAM

- At least 8M bytes of free memory after you have loaded your Windows interface and any other applications besides the SLD software.
- At least 5M bytes of available disk space
- A VGA or Super VGA graphics card and color monitor (a graphics accelerator card recommended to boost performance; a monitor capable of at least 800x600 operation recommended)
- A mouse
- A serial port for connection to the emulator (16550 UART recommended for operation at 57.6K baud and above)
- At least 4M bytes for a swap file (permanent swap file recommended, with a disk cache such as smartdrive for improved Windows performance)
- Config.sys entries of at least Files=30 and Buffers=30

Defining the Debug Environment

This chapter describes how to:

- *Create a loadfile for symbolic debugging and emulation.*
- *Invoke and exit the SLD software.*
- *Configure the emulator for your target processor and your personal working style.*
- *Create and run command scripts, including an automatic command script.*

Creating a Loadfile

To debug at the source level (with source code and symbolic names), you must retain symbolic debugging information in your loadfile. Use compiler, assembler, and linker switches to suppress optimization and to add symbolic information. See your toolchain documentation.

Be sure your loadfile is in OMF86 or OMF386. Most x86 toolchains can generate the appropriate format. Contact your toolchain vendor for specific information.

Toolchain CAUTION

The emulators and debuggers are not guaranteed to work correctly with unsupported toolchains.

For information on toolchain options, see the *Hardware Reference* and the readme.txt file.

Starting and Ending an Emulator Session

Power CAUTION

Turn on the emulator before turning on your target system. Power must be applied and removed in the correct sequence. Failure to follow this sequence will severely damage your target system and the emulator. Turn power on in the following sequence:

1. *Apply power to the emulator.*
2. *Apply power to the target system.*

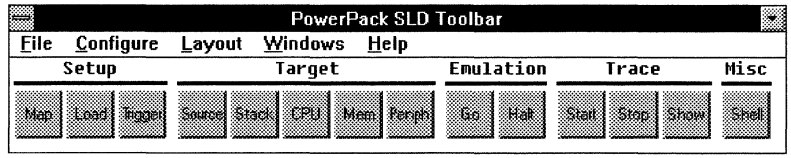
PowerPack SLD

Once the software is installed on your host computer, the firmware is loaded into your emulator, and your target system and the emulator are powered-on, start an emulation session from the PowerPack SLD icon (shown at left). The first time you invoke the SLD software after installation, a series of dialog boxes require initial information.

The Toolbar is the first SLD window to appear and must remain open. Closing the Toolbar exits the SLD software. Minimizing the Toolbar hides any other open (including minimized) SLD windows; restoring the Toolbar redisplays (with the same screen layout) those SLD windows.

Toolbar buttons and menus provide quick access to the most frequently used commands and windows. Grayed-out buttons indicate features unavailable for a particular processor or emulator configuration.

Toolbar: the SLD software's main control panel



Before starting emulation, initialize the emulator for the modules you are debugging and arrange the desktop for your own convenience. Such preliminary tasks can include:

- Start a record of your Shell window activities.
- Map memory and specify some loading options.
- Enable display updates to occur during emulation.
- Enable signals and specify initial CPU and peripheral register values.

You can do many of these tasks with the SLD menus and buttons, from the Shell window command line, or from a script (an ASCII file of Shell commands) in the Shell window. You may also need to edit powerpak.ini with a text editor.

To end an emulator session, do one of:

- Choose the Exit command from the file menu on the Toolbar.
- Double-click the system box in the upper left corner of the Toolbar.
- With focus on the Toolbar, press <Alt><F4>.

Power CAUTION

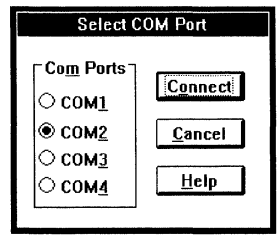
Turn off your target system before turning off the emulator. Power must be applied and removed in the correct sequence. Failure to follow this sequence will severely damage your target system and the emulator. Turn power off in the following sequence:

1. *Remove power from the target system.*
2. *Remove power from the emulator.*

Selecting a COM Port and Baud Rate

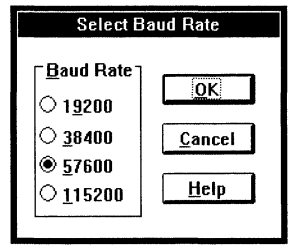
If your emulator is connected to your host PC via RS-232C serial communications and you are starting the SLD software for the first time since installation, you must specify the COM port and baud rate used for communication between your host system and the emulator. Your choices are saved in powerpak.ini. In the Select COM Port dialog box, choose the appropriate serial port and choose Connect.

Select COM Port dialog box for serial communication between your PC and emulator



In the Select Baud Rate dialog box, choose the appropriate baud rate. On some host systems, baud rates above 57600 can require a special Windows driver.

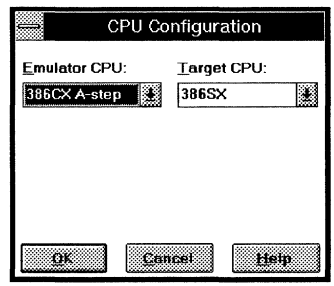
Select Baud Rate dialog box for communication between your PC and emulator



Co-ordinating Intel386 Emulator and Target CPUs

For an Intel386 emulator, a CPU Configuration dialog box appears the first time you start the SLD software. (If you first see a message box asking you to remove a jumper, ensure there is no jumper on TP1.)

CPU Configuration dialog box for co-ordinating the emulator's bondout processor with your target processor



In the Target CPU field, select the processor in your target design. In the Emulator CPU field, select the stepping of the bondout processor in the emulator probe head. To discover the stepping, look for the part number (FPO) on the chip. Production FPOs are 8 digits followed by a change indicator. Pre-production and obsolete parts use a 5-digit code starting with Q.

CPU	Step	Production FPO	Pre-production FPO
386EX	A	xA or xB	Q8492
	B	xD	Q7949
	C		Q8042
386CX or SX	A	xA	Q8307
	B	xB	Q8543

Starting a Log File

A logfile records all that appears in the Transcript pane of the Shell window. The following sample sequence sets up the Transcript pane and opens a log file to record Shell commands and results.

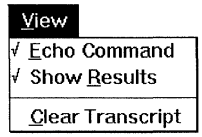
Sequence of Shell commands for logging

```
Echo On;                    // Commands you enter appear
                            // in the Transcript pane.
Results On;                 // Results of the commands appear
                            // in the Transcript pane.
DasmSym On;                 // Disassembly in the Transcript
                            // pane uses symbol names.
Log "emu1.log";             // The log filename is emu1.log.
Overwrite;      // Each time you start logging overwrites any prior
                // logging. The opposite command is Append
Logging On;     // Start writing to emu1.log. The date and time
                // are recorded when you start and stop logging.
Version;                    // Display and log version information for
                            // the emulator, DOS, and Windows.
//...                       // Your emulation session activities...
Logging Off;                // Stop writing to emu1.log.
```

You can do some of the above commands in the Shell window menus

- To echo commands, toggle the View menu Echo Command item.
- For results display, toggle the View menu Show Results item.

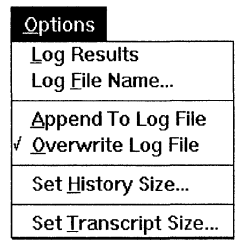
Shell window View menu with Echo Command and Show Results enabled



- To specify whether to overwrite or append new information to an existing log file, choose the Options menu Overwrite Log File item or Append To Log File item.
- To specify the log filename, fill-in the Options menu Log File Name dialog box.
- To start or stop logging, toggle the Options menu Log Results item.

The next time you start logging, the new log overwrites any previously logged information, destroying the logfile's previous contents.

Shell window Options menu with Log Results disabled (logging is stopped) and Overwrite Log File enabled



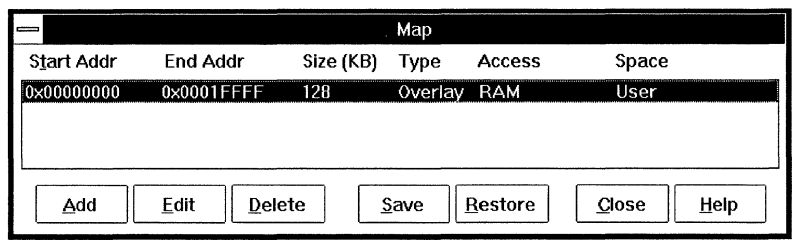
Mapping and Initializing Memory

This section applies to emulator configurations with overlay memory.

Before loading your code or symbols, you must map memory. You can use a memory map saved previously or specify a new configuration.

Open the Map dialog box from the Toolbar either with the Map button or by choosing the Configure menu Map item. The following shows a Map dialog box with 8K bytes mapped.

Map dialog box with 128K bytes of overlay memory mapped for RAM (unrestricted read and write) access



The Map dialog box lists any already configured sections of memory. Use the buttons along the bottom of the Map dialog box to:

Add Configure a new section of memory.

Edit — Reconfigure the selected section. Use the mouse or arrow keys to select from the list in the dialog box.

Delete — Revert the selected section to unconfigured memory.

Save — Save to a map file the memory configuration listed in the dialog box.

Restore — Configure memory from a previously saved map file.

The Add and Edit buttons pop-up a dialog box to specify regions as:

- for PP-386 and SW-386 emulators, any multiple of 4K bytes starting on any 4K address
- for EA-486 emulators, any multiple of 128K bytes starting on any 128K address
- for EA-NS486 emulators, any multiple of 64K bytes starting on any 64K address

You can specify the size either as a hexadecimal number of bytes with the Length button selected or by a hexadecimal ending address with the End Addr button selected.

The Add and Edit dialog boxes also provide mapping options, with inapplicable options greyed-out depending on the target processor:

- overlay or target memory, as listed in the Map dialog box Type column
- for 386 EX, 386 CX, and Intel486 SLE processors, User or SMM space, as listed in the Map dialog box Space column
- how the emulator treats memory accesses, as listed in the Map dialog box Access column:

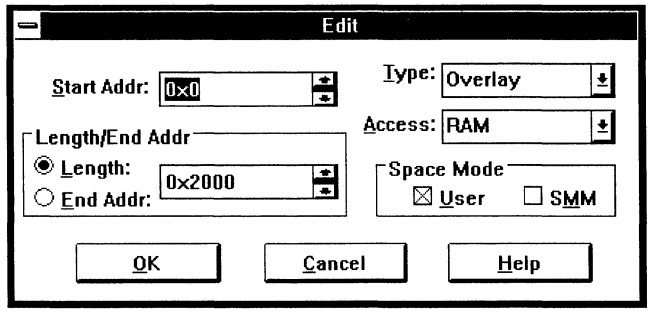
  RAM — allows reads and writes without breaking.

  ROM break — allows reads; disallows writes; an attempted write causes a break. For 386 and Intel486 emulators with memory mapped to Target, writes are allowed but break emulation. This option is unavailable for EA-NS486 emulators.

  ROM nobreak — allows reads; disallows writes; does not break on any access. For 386 and Intel486 emulators with memory mapped to Target, ROM nobreak is the same as RAM; that is, writes are allowed and do not break emulation.

  NONE — disallows reads and writes; breaks on any access. For 386 and Intel486 emulators with memory mapped to Target, accesses are allowed but break emulation. This option is unavailable for EA-NS486 emulators.

Edit dialog box, accessed from the Map dialog box Edit button; similar to the Add dialog box popped-up from the Map dialog box Add button



You can also use the Shell window to map memory. The following sample sequence of commands prepares a 386 emulator and memory for loading code or symbols:

Mapping: Shell command sequence

```
Map Clear;                    // Maps all memory to target, removing
                              // any existing map configuration.

RestoreMap "emu1.map";        // Maps memory from a map saved
                              // previously. emu1.map contains
                              // the line: map 0x0 0xffff ram.

Map 0x10000 RomBrk;           // emu1.map maps only part of memory,
                              // not including the 4K-byte block starting
                              // at address 0x10000. This Map command
                              // configures memory from 0x10000 to 0x10fff
                              // as ROM and specifies that any attempt to
                              // access this space will break emulation.
```

Loading a Loadfile

Once memory is configured, you can load the file to be debugged. The PowerPack emulators support OMF86 and OMF386 loadfile formats.

For loadfiles generated with the Borland C compiler, before loading enter MaxBitFieldSize 16 on the Shell command line.

You can load a file during emulation. Be sure the file's load addresses do not overlap the memory occupied by the running program. Loading a file at a location in use stops the emulator in an unpredictable state.

The following sample sequence of commands loads code and symbols:

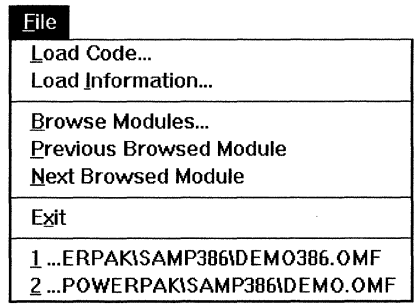
Loading: Shell command sequence

```
Loadsize Long;          // (default) The loadfile is written to memory
                        // in double-word accesses, which is the
                        // fastest way to load code.

Load "myfile.obx" code symbols nodemand nowarn status;
                        // Load code and symbols from the myfile.obx loadfile.
```

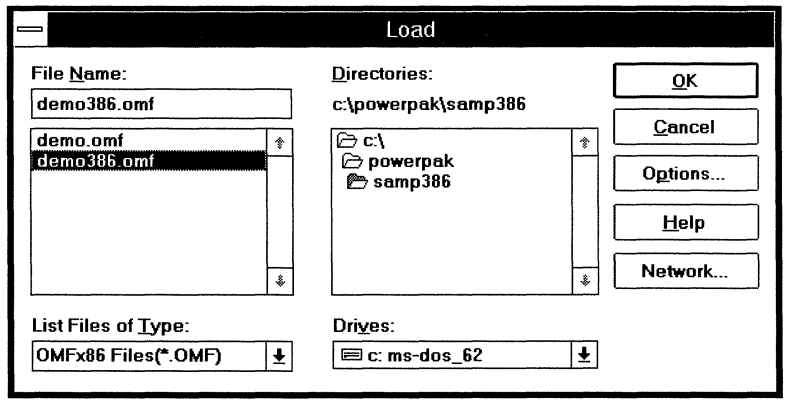
You can do the above operations using various SLD window menus. To load code and symbols, open the Load dialog box with the Toolbar Load button or with the Source window File menu Load File item. To reload one of the last four files loaded, you can choose a Source window File menu loadfile pathname. The pathnames are added to the bottom of the File menu as you load files.

Source window File menu showing the two most recently used loadfiles



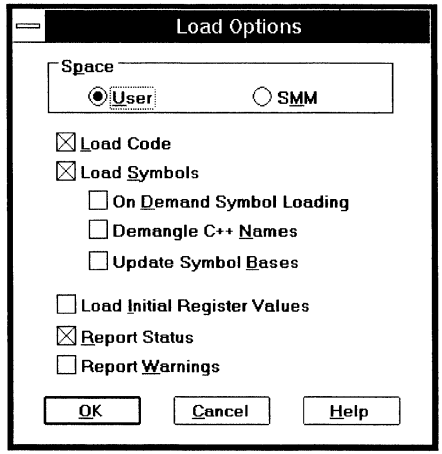
In the Load dialog box, the name of the previous file that was loaded is automatically filled-in. Or, you can browse the directory and file lists to specify a different loadfile.

Load dialog box, accessed from the Toolbar Load button



Before choosing the OK button to load the file, you can choose the Options button in the Load dialog box to open the Load Options dialog box. The loadfile format (OMF86 or OMF386) and the target processor determine what options are available; some options may be missing or greyed-out on your emulator. If you have already loaded a file, the options you specified previously are preserved.

Load Options dialog box, with options for loading an OMF386 loadfile into a 386 EX emulator, popped-up from the Load dialog box Options button



Be sure the space option (User or SMM) you select is compatible with the address space you configured in the Map dialog box. This option is applied to where the code is loaded.

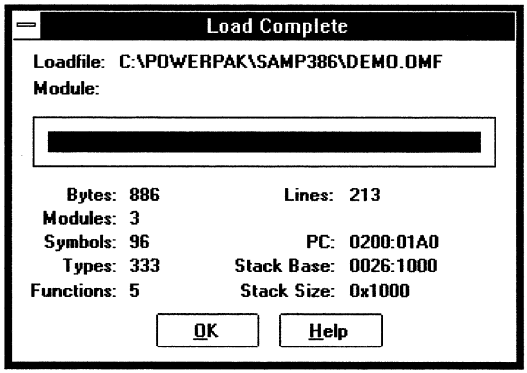
You can load code, symbols, or both from any loadfile. For example, load only code if symbols are already loaded; load only symbols for debugging ROM code. To load code, check the Load Code box. To load symbols, check the Load Symbols box and any combination of boxes under Load Symbols:

- On-demand symbol loading defers loading local symbol and line-number information for each module until it is needed; i.e. until either the module is displayed in the Source window or a breakpoint is set in the module. Advantages of on-demand symbol loading include faster initial loading, faster lookup for the symbols that are demanded, and less memory occupied by the loaded file because only the fewest required symbols are loaded.
- For C++ code containing virtual functions, overloaded functions, and some other symbol types, the emulator can demangle the first instance of each such symbol. Subsequent instances remain mangled in the emulator symbol table rather than duplicated, so you can access all symbols in your program. However, the names do not appear mangled in your source. The warning message C++ duplicate name detected alerts you to the presence of mangled names.
- OMF386 symbol server base addresses can be updated in conjunction with register initialization.

OMF386 startup code or linker directives can initialize the processor registers.

You can request or suppress information about the load process and results. For a dynamic report of the loading process, check Report Status. In the Load Complete dialog box, a bar graph fills to indicate the percent loading complete; loading statistics are updated continuously during the load process. To review the load information after closing the Load Complete dialog box, open the Source window File menu Load Information dialog box.

Load Complete dialog box, similar to the Load Informaton dialog box, showing the results of loading an OMF86 loadfile



Suppress warning messages during loading by un-checking Report Warnings.

Symbolic Addresses

Any program symbol, interpreted as a symbolic :<offset>, is a virtual address. You can reference a symbol in a command, dialog box, or expression. Simplify such references by taking advantage of how the emulator resolves names. For example, for a symbol in the current module, you need not specify the module and function.

The loader creates a symbol table with the names of all modules, functions, variables, and line numbers in the loadfile. The symbol information is hierarchical, with each symbol representing a range of addresses that can contain other symbols. At the top of the hierarchy are modules, public labels, and public variables. The subsequent levels are:

Modules contain functions, static variables, and line and column numbers.

Functions contain parameters, local variables, static variables, line numbers, and blocks.

Blocks are handled as unnamed functions. Nested blocks can contain local and static variables defined in scope.

Using this symbol hierarchy, you can uniquely specify any symbol. Fully qualified symbols have one, two, or three alphanumeric names beginning with #. Partly qualified symbols default to the current module and function, that is, the scope of the current program counter.

1. Look up the symbol at the lowest level of the hierarchy.
2. If no match is found, look up the symbol at the next level.
3. If no match is found, look up the symbol at the global level.
4. If no match is found, the symbol name does not exist. Return a symbol-not-found error.

To find the address of a symbol with one name:

- If the module and function are defined by the context, look up the name as a variable within the scope of the function.
- If the module but not the function is defined by the current context (for example, you have stepped from the module into a called assembly routine), look up the name within the scope of the module.
- If no module or function is defined by the current context, look up the name as a module, public variable, or label.
- If the name is a number, look up the number as a module name or as a line number within the current module.

One-name symbols

#module1	Returns the beginning address of module1.
#function1	For a function in the current module, returns the address. Otherwise, returns the address of a function in the global table. (Only static functions are not in the global table.)
#variable1	Returns the address of a global or public variable or of a variable inside a nested block, function, or module.
#55	Returns the address of line 55 in the current module.

To find the address of a symbol with two names:

- If a module is defined by the current context, look up the first name as a function contained within the module. Otherwise, look up the first name as a module, then as a global function.
- If the module and function are defined by the context, look up the second name as a variable within the scope of the function.
- If the module but not the function is defined by the current context (for example, you have stepped from the module into a called assembly routine), look up the second name as a variable within the scope of the module.

- If no module or function is defined by the current context, look up the second name as public variable or label.
- If the first name is a number, look up the first name as a module name or as a line number within the current module. If the second name is a number, look up the second name as a line number if the first name is a module or function, otherwise as a column number.

Two-name symbols

#55#15	Returns the address in the current module on line 55, column 15.
#module1#100	Returns the address of line 100 in module1.
#module1#func1	Returns the address of func1 in module1.
#module1#var1	Returns the address of var1 in module1.
#func1#var1	Returns the address of func1 in the current module. Or, if func1 is global, returns the address of var1 in the scope of func1.

To find symbolic variables with three names:

- The first name must be a module. The second and third names can be line and column numbers in the module; or, the second can be a function in the module while the third is a variable or line number in the second's scope.
- If the third name is a variable it is first looked up within the module and function context. If not found, it is looked up as a global variable or label. A globle symbol's address is returned even if outside the scope of the module identified by the first name.

Three-name symbols

#mod1#25#1	Returns the address of module mod1, column 1, line 25.
#mod1#func1#100	Returns the address of module mod1, line 100.
#mod1#func1#var1	Returns the address of module mod1, function func1, variable var1.

To display line numbers in the Source window, open the View menu and check Line Number. In the Shell window, you can list all line-number records for the current module with displaySymbols lines.

Some line numbers are comment lines and have no compiled code.

Enabling Memory Access

You can access memory during emulation, to read or write the current values in target memory and on-chip peripheral registers (but not CPU

registers). Such reads and writes take a small, additional amount of processor time and can thus affect your program's performance. Memory access is initially disabled and must be enabled if, for example, you want to refresh the Memory or Peripheral window during emulation. To enable memory access, either:

- On the Shell command line, enter RunAccess On.
- Enable (check) the Toolbar Configure menu Run Access item.

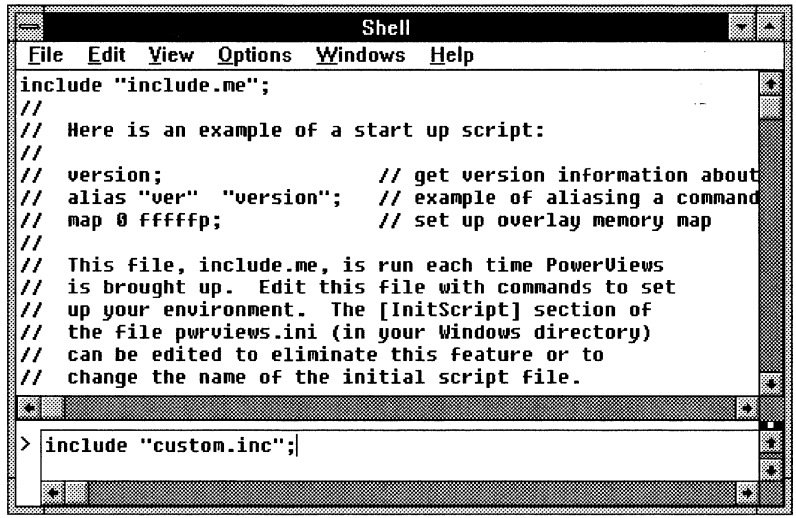
Run Access does not allow CPU register access. The CPU registers cannot be accessed during emulation; their display is updated only when emulation halts.

Using a Script

A script is a text file of Shell commands. To run a script, use the Include Shell command or the Shell window File menu Include File dialog box. You can put an Include command in a script.

In the powerpak.ini file [InitScript] section, you can specify a script to run automatically at SLD initialization. Edit the script = line in powerpak.ini. For example, script = c:\sld\user\myscript. If you specify no pathname (for example, script = myscript), be sure your script is in the directory with the SLD software.

Shell window after the include.me sample initialization script has run, with an Include command to run custom.inc ready to be entered on the Shell window command line



Leveraging Previous Emulation Sessions

You can shorten your setup time in subsequent emulation sessions by saving map, chip select, event, and log files.

You can save the map information to a file. In the Shell window enter a MapSave command, specifying a path and filename; or, fill-in the Map dialog box Save button dialog box. Later, you can restore the saved map with a Shell window MapRestore command or the Map dialog box Restore button.

You can save chip select information. In the Shell window enter the SaveCS command, specifying a path and filename; or, fill-in the Toolbar Configure menu Save Chip Selects dialog box. Later, you can restore the saved registers with the Shell window RestoreCS command or the Toolbar Configure menu Restore Chip Selects item. See the *Hardware Reference* for a list of the registers saved for each processor.

You can save event definitions. In the Shell window enter an EventSave command, specifying a path and filename; or, fill-in the Event window File menu Events As dialog box. Later, you can restore the saved events with the Shell window EventRestore command or the Event window File menu Restore Events item.

Instead of retyping command sequences, you can save the sequence to be made into a script that you can run with an Include command or automatically as the initialization script. During an early emulation session, even if you usually use the menus, open a log file and record lengthy or frequently repeated tasks by entering the commands in the Shell window. Edit the log file with a text editor, creating a script to be run in future emulation sessions. By logging an emulation session, you can test and record error-free command sequences.

Keyboard Shortcuts

You can use function keys instead of commands or menu items:

F1	Open a window for SLD on-line help.
F2	Halt emulation.
F3	Start trace.
F4	Stop trace.
F5	Set focus to the Toolbar window.
F6	Set focus to the next open SLD window.
F7	Step Into.

F8	Step Over.
F9	Start emulation (Go)
F10	Activate the menu bar for keyboard use.

Example: Enabling Intel386 EX Expanded Memory

You can read and write any peripheral register by editing the field values in the Peripheral window or by entering Dump, Fill, and Write commands on the Shell window command line.

To access some of the peripheral registers with the Shell commands, you must first enable expanded I/O space. Once expanded I/O space is enabled, you can use both the Peripheral window and the Shell command line to access many peripheral registers.

When expanded I/O space is disabled, the affected registers appear in the Peripheral window with question marks (?) in their address fields. A question mark indicates you can access the register via the Peripheral window but not from the Shell command line.

To enable expanded I/O space, **close (not minimize) the Peripheral window**, then set the ESE bit in the REMAPCFG register by three sequential writes to I/O addresses 0x22 and 0x23. (The sequence must write twice to each address.) For example, enter the following Size and Fill commands on the Shell command line:

```
Size Byte;
Fill 23p 23p 0x00 Byte IO;
Fill 22p 22p 0x80 Byte IO;
Size Word;
Fill 22p 23p 0x0080 Word IO;
```

The Size command specifies the physical size of the data access. The Byte and Word specifiers in the Fill commands inform SLD of the supplied data format.

Debugging in Source and Stack

This chapter describes how to:

- *Set, view, and clear breakpoints.*
- *Control program execution.*
- *Examine and modify variables and the stack.*

Viewing Source

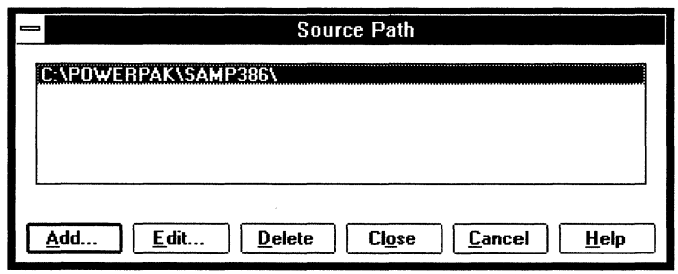
After loading an executable file, you can view the source file associated with each module in the Source window. The Source window initially displays code at the current program counter (CS:EIP). The instruction or statement pointed to by the program counter is marked by >>.

When you open the Source window after loading but before executing code, the program counter may be in the assembly startup code designed to execute before main(). If the startup code is from an available assembly source file, the Source window displays the assembly source. If the startup code was generated by the compiler, the Source window displays the disassembly from memory.

To view a different module, choose the File menu Browse Modules item. All loaded modules are listed. If a module's source has been modified more recently than the loadfile, a warning message appears and an asterisk marks the source filename in the Source window title.

If the emulator cannot find the source file corresponding to the module you are browsing, you may need to modify the source search path list. Modify the list in the Source window Options menu Source Path dialog box.

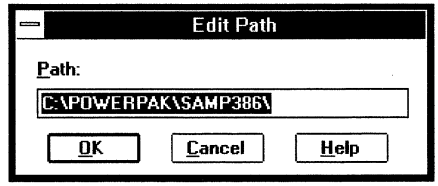
Source window Options menu Source Path dialog box



To add a pathname to the Source Path dialog box, choose the Add button and enter a directory or file pathname in the Open dialog box.

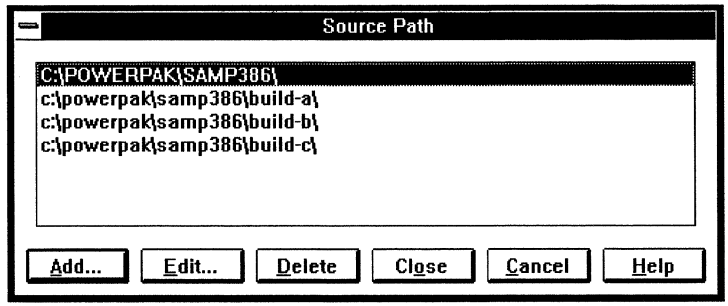
To edit a path, use the mouse or the <Up Arrow> and <Down Arrow> keys to select a path in the Source Path dialog box; choose the Edit button; and edit the path string.

Edit Path dialog box, accessed from the Source window Options menu Source Path dialog box Edit button

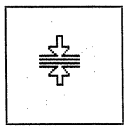


The emulator searches the paths in the order they are listed in the Source Path dialog box, stopping at the first file that matches the source filename in the loadfile. If you have duplicate filenames in different directories, order the source path search list so the emulator finds the correct one first. For example, in the following, the emulator searches first samp386, then build-a, build-b, and finally build-c.

Source window Options menu Source Path dialog box listing multiple paths to be searched sequentially from top to bottom of the list



When symbolic information (including the source file pathname) is available for a module, you can view the module as source code with or without interleaved disassembly. Use the Source window View menu to toggle between Source Only and Mixed Source And Assembly. Modules with no associated source file can only appear as disassembly. To see symbols in the disassembly, check the Toolbar Configure menu Symbolic Disassembly item.



You can split the Source window into two panes by clicking and dragging on the split box at the top of the vertical scroll bar. A split-box cursor appears at the right of the split bar (see figure at left). To resize the panes, use the mouse to drag the split box.

With two Source window panes, you can work in two different modules or two areas of the same module independently. To move between panes, click in the inactive pane to make it active.

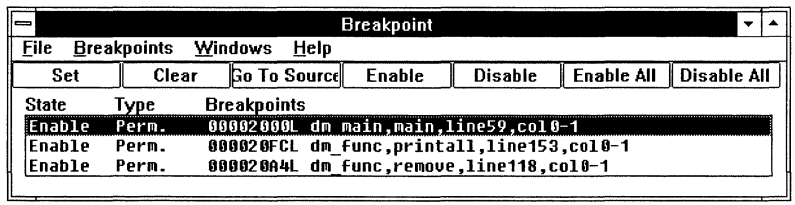
Managing Breakpoints

At a breakpoint, emulation halts before executing the instruction at the breakpoint address. A temporary breakpoint is then cleared; a permanent breakpoint remains. A breakpoint set on a non-executable statement automatically moves to the next executable instruction.

In PP and EA emulators, you can set 256 software breakpoints; in SW emulators, you can set 128 software breakpoints. You can set up to four hardware breakpoints, which use the DR[0:3] debug registers. See the DR command description in the "Shell Window Reference" chapter.

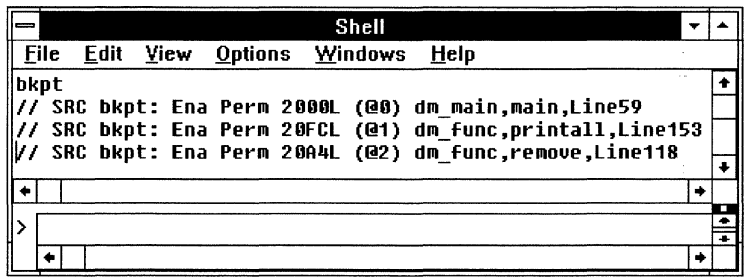
To display the currently set breakpoints, open the Breakpoint window.

Breakpoint window listing the state (enabled or disabled), type (permanent or temporary), and source location of each currently defined breakpoint



To list breakpoints in the Shell window, enter a Bkpt command with no arguments.

Shell window showing breakpoints listed in response to a Bkpt command



You can set breakpoints from the:

- Shell window Bkpt command
- Breakpoint window Set button or Breakpoints menu Set Breakpoint item
- Source window source display or various Breakpoints menu items

To set a breakpoint from the Source window display, using the mouse:

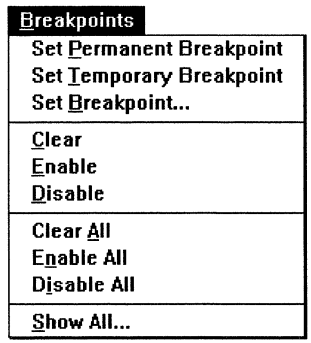
1. Move the mouse pointer to the left of the source line where you want to set a breakpoint.
2. When the mouse pointer changes shape to a cross-hair cursor (shown at left), click on the primary mouse button to set a permanent breakpoint or on the secondary button to set a

temporary breakpoint. (On a mouse configured for right-handed use, the primary is the left button and the secondary is the right button.) The line with the breakpoint is highlighted in red.

Alternatively, using the Source window Breakpoints menu, either:

- Position the Source cursor where the breakpoint is to be set and select Set Permanent Breakpoint or Set Temporary Breakpoint.
- Regardless of the Source cursor position, choose Set Breakpoint and fill-in the Set Breakpoint dialog box.

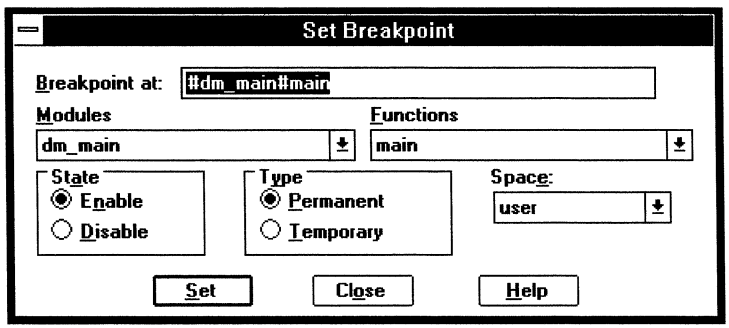
Source window Breakpoints menu



To set a breakpoint from the Breakpoint window, pop-up the Set Breakpoint dialog box from either the Set button or the Breakpoints menu Set Breakpoint item.

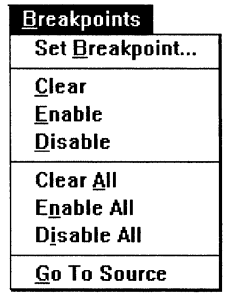
The Set Breakpoint dialog box Breakpoint At field accepts both numeric and symbolic addresses. For symbolic addresses, you can browse the Modules and Functions drop-down lists. For C++ source, mangled names (which you can also list with a DisplaySymbols Shell command) appear in these drop-down lists. These names include member functions from all classes defined in a source module and its header files; global (non-class related) functions; and compiler-provided default constructors and destructors.

Set Breakpoint dialog box, accessed from either the Source window Breakpoints menu Set Breakpoint item, the Breakpoint window Breakpoints menu Set Breakpoint item, or the Breakpoint window Set button



You can co-ordinate the Source and Breakpoint window displays. To open the Breakpoint window from the Source window, choose the Source window Breakpoints menu Show All item. To display the source of a specific breakpoint, in the Breakpoint window highlight the breakpoint and choose either the Breakpoints menu Go To Source item or the Go To Source button.

Breakpoint window Breakpoints menu



Avoid setting breakpoints on inline functions. The Set Breakpoint dialog box flags no inline functions. If you have set a breakpoint on a function and stepping does not advance the Source window cursor, it is an inline function. Stepping through instructions in your class definition advances the program counter but not the Source cursor. Remove the breakpoint on the function and restart emulation.

In Mixed Source And Assembly view, the assembly instructions for all inline functions appear after the last source line of the module.

If your program has more than one source statement per line number and the toolchain provides statement-level line number information, you can set a breakpoint on any statement in a line. For example:

Set a breakpoint: multiple statements per line

```
If (errorNumber) errorHandler(errorNumber);
```

To set a breakpoint on the errorHandler call, when errorNumber is nonzero:

1. From the Source window Options menu, set the level of step granularity by toggling Step Execution Granularity to Statement.
2. Click on errorHandler(errorNumber), open the Breakpoint menu, and choose Set Permanent Breakpoint. Or, double-click on errorHandler(errorNumber) and choose Permanent Breakpoint.
3. The entire line is highlighted as a breakpoint, with the actual breakpoint set on the second statement. From the View menu, choose Mixed Source And Assembly to see the breakpoint on the second statement.

To set a breakpoint at the statement level, you must know how many spaces your compiler uses for a tab character. For example:

How tab width affects setting breakpoints at statement level

The following line of three statements is compiled with a tab width of eight:

<tab><tab>for (j = 0; j < max_num; j++) {

The compiler tab width produces the following column ranges:

j = 0;	columns 0 through 26
j < max_num;	columns 27 through 39
j++	columns 40 through 45

Setting the Source window tab width to four instead of eight puts j = 0; at column 13 and j < max_num; at column 20. It is then difficult to set a breakpoint on the correct statement.

Symbols must be loaded before you can set breakpoints on line numbers or functions. If you chose On Demand Symbol Loading when loading your program, the symbols needed for a breakpoint are loaded either when you set the breakpoint or when you display the source for the module containing them.

You can enable and disable all or individual breakpoints, using either the Source or Breakpoint window Breakpoints menu Enable/Disable (All) items or the Breakpoint window Enable/Disable (All) buttons. An enabled breakpoint is defined and active; emulation breaks when the breakpoint is reached. A disabled breakpoint is defined but inactive; emulation does not break when the breakpoint is reached.

Using disabled and enabled breakpoints

For example, an interrupt handler named MyIntr (in a module named ModB) might be started at any time. To discover whether MyIntr is starting during execution of another function named Atomic (in a module named ModA), the designer does the following:

1. Set a breakpoint, enabled, at the beginning of #ModA#Atomic.
2. Set a breakpoint, enabled, at the end of #ModA#Atomic.
3. Set a temporary breakpoint, disabled, at #ModB#MyIntr.
4. Go. MyIntr can execute without causing a break.
5. At the first Atomic breakpoint, enable the MyIntr breakpoint. Calling MyIntr during Atomic execution causes a break and clears the MyIntr breakpoint. If MyIntr is not called, at the second Atomic breakpoint disable the MyIntr breakpoint.

You can remove all or individual breakpoints by any of:

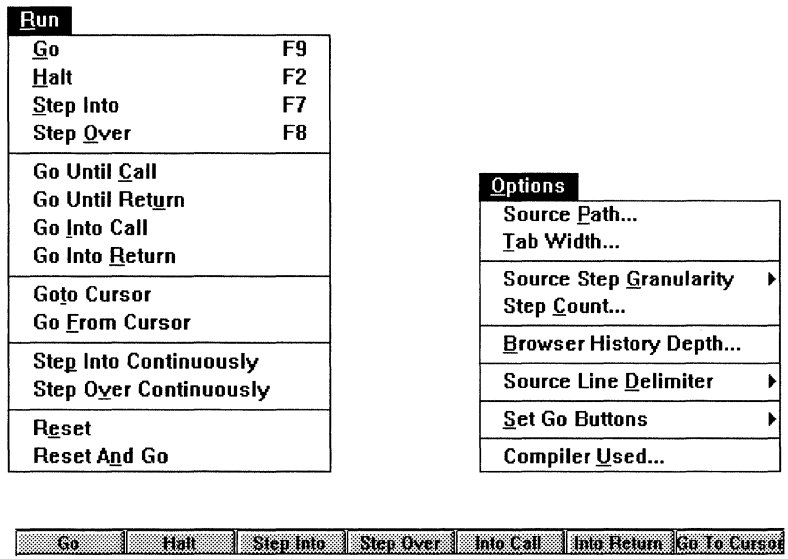
- Choose the Source or Breakpoint window Breakpoints menu Clear All item.

- In the Breakpoint window, select a breakpoint and choose Clear from either the buttons or the Breakpoints menu.
- In the Source window, click in the left margin of the red-highlighted line containing the breakpoint; or, move the cursor to the breakpoint and choose the Breakpoints menu Clear item.
- On the Shell command line, enter a BkptClear command.

Starting and Stopping Emulation

With the Source window buttons and menus and various Shell commands, you can emulate one or more instructions at a time or as a free-running program.

Source window Run and Options menus and button bar



Step breaks after executing one to 100 instructions or statements, according to how you set the Options menu Step Count and Source Step Granularity items. The Shell Step and StepSrc commands can do the same.

Step Into and Step Over specify how transfer instructions (such as jumps or function calls) affect where emulation breaks after stepping:

Into	breaks at the first instruction or statement at the transfer destination.
Over	breaks at the first instruction or statement following the transfer instruction.

Continuously — repeatedly steps until you halt it.

Go — executes your program to the next enabled breakpoint or until you halt it. The Toolbar Go button and the Go Shell command do the same. The GoInto and GoUntil Shell commands provide the same functionality as the Go Until/Into Call/Return buttons and Run menu items.

- From Cursor — moves the program counter to the instruction at the Source cursor, then starts emulation.
- To Cursor — emulates until the program counter reaches the Source cursor.
- Into Call — breaks at the first instruction or statement at the next transfer destination.
- Into Return — breaks at the first instruction or statement following the next transfer instruction.
- Until Call — breaks at the last instruction or statement before the next transfer instruction.
- Until Return — breaks at the last instruction or statement before a return from the next transfer instruction.

To change the Into Call and Into Return buttons to Until Call and Until Return, select from the Options menu Set Go Buttons item Until Call/Return choices.

Reset And Go — Resets your target system, then operates as Go. The ResetAndGo Shell command does the same.

Halt — Stops emulation. The Toolbar Halt button and the Halt Shell command do the same.

To discover whether emulating or halted, look in the Status window or icon or enter EmuStatus on the Shell command line. When emulation has halted, to discover the cause of the break, look in the Status window or enter Cause on the Shell command line.

How fast a Step operation executes depends on the number of SLD windows open. Each window must be updated after each step. You can close or minimize any open SLD window (except the Toolbar) to improve performance. Speeding up stepping can be useful when you use long or frequent Step Continuously operations.

In C++, stepping into a declaration can call a constructor with any initialization parameters and its base class constructors.

Examining Source After Emulating

The Source window display shows the next statement or instruction:

- When emulation halts at a breakpoint, the program counter stops at the instruction containing the breakpoint.
- When emulation halts after a Step Into or Go Into Call, the program counter points to the first instruction in the function.
- When emulation halts after a Step Over or Go Into Return, the program counter points to the first instruction after the return.
- When emulation halts after a Go Until Call or Go Until Return, the program counter points to the call or return instruction.

In Source Only view, a function with no associated source is not displayed after a Step Into when the program counter points to the first instruction in the function. To display the disassembly of such a function, toggle the view to Mixed Source And Assembly.

You can also view disassembled instructions in the Memory window or by entering a Dasm command on the Shell command line.

To modify loaded instructions, use the Memory or Shell window as described in the chapter on debugging in registers and memory. Such code patching is reflected in the disassembly shown in the Source window in Mixed Source and Assembly view. Note that the disassembly at the patched addresses no longer matches the source file contents.

For C++, you can select the following mangled or demangled symbols in the Source window:

- Function symbols
- Global variables
- Global class objects
- Local variables and class objects

You cannot select class.memberFunction type objects.

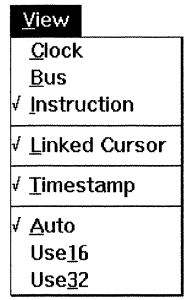
The scope-resolution operator (::) is interpreted as a token separator, not recognized as part of a symbolic address.

Scrolling Trace With Source

When the Source and Trace windows are linked, you can scroll through the Trace window and view the corresponding code scrolling synchronously in the Source window. To link these displays:

1. In the Trace window, open the View menu and choose Instruction to display the trace as disassembly.
2. Re-open the View menu and choose Linked Cursor.

Trace window View menu co-ordinating the Trace and Source window displays

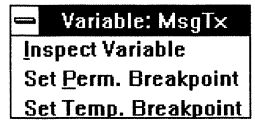


Examining and Editing Variables

You can examine and edit global, static, and local variables in the Variable window by either:

- In the Source window, double-click on the name of the variable you want to view. In the pop-up menu, choose Inspect Variable.

Variable menu, popped-up by double-clicking on a variable named MsgTx in the Source window



- In the Variable window, open the Variable menu, choose Add, and enter the name of the variable you want to view. Specify a fully qualified symbol.

In the Variable window, you can:

View — variable types and values. Non-pointer variables appear in magenta. For enum type variables, the enumerated name follows the hexadecimal value. For example:

```
enum color c = 0x2 = lavender
```

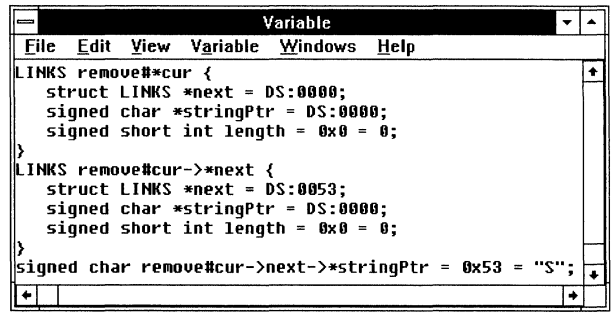
Dereference — a pointer variable by double clicking. Dereferenceable pointers appear in blue. For example, DS:000E is the address of the variable pointed to by cellPtr:

```
CELL_TYPE *printall#cellPtr = DS:000E
```

To dereference a pointer, either double click on the pointer name, or select the pointer and choose the View menu Show item. A new line appears in the Variable window listing the location pointed to. The

following shows a Variable window with next dereferenced from the first entry (cur) and stringPtr dereferenced from the second entry (the dereferenced next):

Variable window showing cascaded dereferenced pointers



Edit — a value. Editable values appear in red. Integer variables can be edited in hexadecimal or decimal, floating point variables in floating point format, and characters in their hexadecimal ASCII equivalent. To edit a value, either double-click on the value; or single-click on the value and choose the Edit menu Edit item. Press <Enter> to end editing or <Esc> to cancel editing. Outside of the current stack context, local variable values are unknown.

Select — a variable or its value by clicking on it. Yellow indicates a selected symbol or value.

Remove — a selected variable from the display. Either choose the Variable menu Delete item or press the <Delete> key. This does not delete the variable from your program, only from the current variable inspection list.

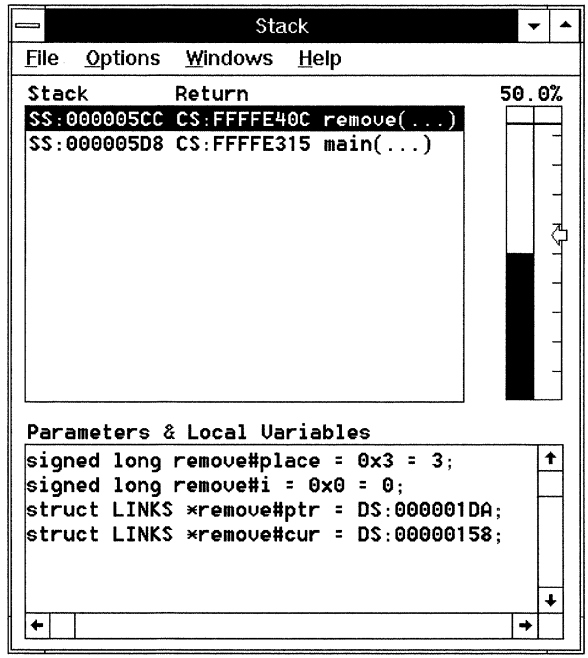
Retrieve — removed variables with the Variable menu Undelete item.

You can also examine program symbolic information using the Shell AddressOf, NameOf, ConfigSymbols, DisplaySymbols, GetBase, SetBase, and RemoveSymbols commands.

Monitoring the Stack

The Stack window contains a stack list pane, a variables list pane, and a stack meter. (You can also list the stack information in the Shell window using StackInfo and DisplayStack commands.)

Stack window with emulation halted in remove, called from main, showing stack usage down to 50.0% from the high-water mark (arrow on the right of the stack meter) of about 58%



Configuring the Stack Window

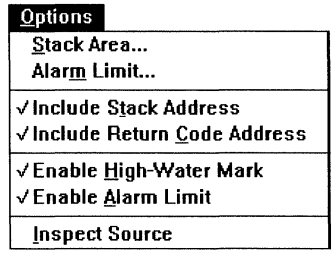
Once a program has executed into one or more functions, the stack list contains frames representing the nested calls. Frame information can include the stack and return addresses of the functions, names of functions with symbolic information available, and the parameters and local variables associated with the function calls. The top frame represents the function currently in scope.

When symbolic information is available for a function, you can display the parameters and local variables in the variables list pane by selecting the frame in the stack list pane. Variables appear in the same format as in the Variable window.

Stack usage is described by the stack meter. The percent of stack area currently in use appears in blue.

To configure the stack and return address display, toggle the Options menu Include Stack Address and Include Return Code Address items. The stack address is the address of the frame in the stack area. The return address is the load address of the next instruction in the calling function.

Stack window Options menu with all stack statistical displays enabled



To view the source or disassembly of a function, select the frame and choose the Options menu Inspect Source item. The Source window displays the function.

You can configure the stack meter to show the highest level the stack has reached since initialization. This high-water mark is an arrow on the right of the stack meter. Enable (check) the Options menu Enable High-Water Mark item; or enter an EnableHighWaterMark Shell command.

You can set an alarm on the stack meter to notify you when stack usage exceeds a specified percentage of the stack area. If the alarm limit is exceeded when emulation halts, a warning message appears. Choose the Options menu Alarm Limit item and specify a percent value from 1 to 100. Then, enable (check) the Options menu Enable Alarm Limit item. Alternatively, in the Shell window you can set an alarm limit and enable the alarm message with SetStackAlarm and EnableAlarmLimit commands. The alarm limit appears as a red line across the stack meter.

No alarm message appears until emulation halts. During emulation, the stack can exceed the alarm limit without displaying the warning message. To monitor the amount of memory used by the stack while emulation continues, emulate by stepping continuously. Choose the Source window Run menu Step Over Continuously or Step Into Continuously item.

Halting emulation updates the stack information with the:

- current function and variable information
- percentage of the stack in use
- High-Water Mark, if enabled
- alarm, if enabled

If, after emulation halts, the monitored stack area is discovered to be mismatched with the program's stack area, some Stack window features are invalidated and grayed-out in the menus. For example, the alarm, high-water mark, and stack meter become unavailable.

Monitoring multiple stacks

For multiple stacks, you can track the stack currently in use. Create Shell aliases to define the base and size of each stack. For example:

```
alias "s1" "SetStackArea 4000 100";
alias "s2" "SetStackArea 3000 100";
```

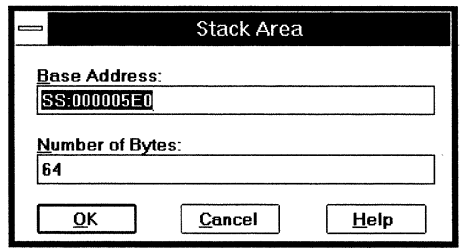
When emulation halts, switch to monitoring the current stack by entering one of the aliases on the Shell command line.

Setting the Stack Base Address and Size

The stack base address and the stack size are typically put into the loadfile by your compiler. Otherwise, the emulator looks for a default stack base address in the powerpak.ini file. If powerpak.ini also specifies no base address, the current stack pointer (SS:ESP) value is used. An undefined stack size defaults to 4K bytes.

To discover the current stack base and size, either enter StackInfo on the Shell command line, or in the Stack window open the Options menu and choose Stack Area. The values in the dialog box describe the current stack allocation. The following shows a Stack Area dialog box.

Stack Area dialog box, accessed from the Stack window Options menu



If you edit these values, ensure the Base Address matches your program's stack base and the Number of Bytes accommodates as much of your program's allocated stack area as you want to watch. When the SS:ESP is outside the stack area recognized by the emulator, the stack statistical information is invalid.

Changing the stack size recognized by the emulator does not affect the amount of memory available to your program for stack activity. Changing the stack base recognized by the emulator does not affect the SS:ESP. The stack base and size are used only by the emulator to maintain the stack usage statistics.

You can also change the stack area by a SetStackArea Shell command or by SetStackBase and SetStackSize Shell commands.

Determining how large a stack area to allocate

The Stack window can help you determine the minimum amount of memory to allocate for the stack:

1. Open the Options menu and choose Enable High-Water Mark.

2. Execute your program for maximum code coverage.
3. Halt execution.
4. Note the high-water mark (maximum stack usage as a percentage of the allocated stack area) on the stack meter.
5. Remake your loadfile, increasing or decreasing the allocated stack for efficient usage.

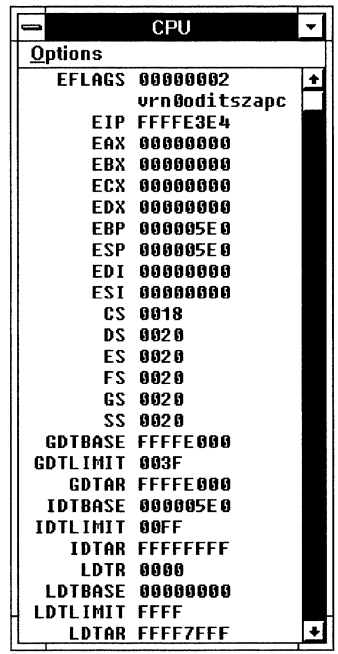
Debugging in Registers and Memory

This chapter describes how to access the CPU registers, the peripheral registers, and memory.

Viewing and Modifying the CPU Registers

You can view and change CPU registers and control signals from the CPU window, Toolbar, Source window, and Shell command line.

CPU window showing the execution point (CS:EIP) at 18:FFFFE3E4 and the stack top and base (SS:ESP and SS:EBP) at 20:5E0



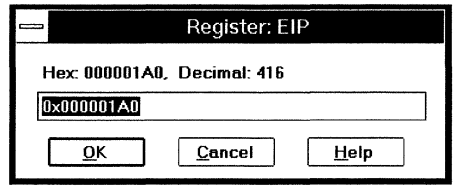
The CPU window is updated when emulation halts. A highlight indicates a register value has changed.

Editing the CPU Registers

To edit a CPU register, either:

- Enter a Register command on the Shell command line.
- In the CPU window, double-click on the register, or select the register and press <Enter>. Enter the new value in the dialog box.

CPU Register dialog box for editing the EIP, popped-up from the EIP line in the CPU window



Resetting the CPU Registers

When you reset and reinitialize the processor:

- The processor RESET pin is asserted.
- The program counter (CS:EIP) is set to 0:FFFFFFF0 for the EA-NS486 and to F000:FFF0 for all other emulators.
- All SLD windows are updated. The Stack window display is invalid because the stack is reset. The Source window displays the beginning of your startup code, at the program counter.

You can reset the processor from the Toolbar Configure menu, from the Source window Run menu, from the CPU window Options menu, or by entering Reset on the Shell command line.

If the reset fails:

1. From the Toolbar Configure menu or the CPU window Options menu, choose Reset CPU Only; or enter Reset CPUonly on the Shell command line. This resets the processor without updating the SLD windows.
2. Reset your target.
3. Reset the processor again, without specifying CPU only, to update the SLD windows.

Resetting the Target Board

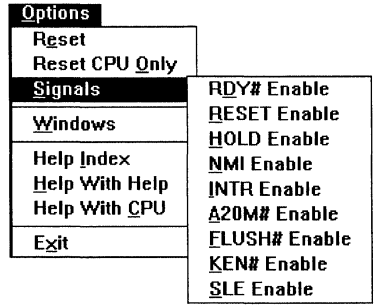
You can reset your target board independently of resetting the SW or EA emulator. To use this feature, connect your target reset input to the Reset Out pin on the back panel of the emulator before turning-on the emulator; and edit powerpak.ini before starting the SLD software. See the Reset command description in the "Shell Window Reference" chapter and the [SystemInfo] section description in the "powerpak.ini File Reference" chapter.

Enabling the Target Signals

Enabling a signal uses that signal from your target system rather than from the emulator. To enable or disable the target signals, check or

uncheck each signal in the CPU window Options menu Signals item. (For a list of configurable signals, see the *Hardware Reference*.)

CPU Options menu Signals configurable for the EA-486



Disabling a signal disconnects it from the target and controls it from the emulator. For example, the emulator drives the 386 signals as:

READY#	asserted
RESET	negated
HOLD	negated
NMI	negated
INT0-INT3 (Intel386 EX)	negated
INT4-INT7 (Intel386 EX)	negated
NA#	negated
SMI# (Intel386 CX and EX)	negated
INTR	negated
A20M# (Intel386 CX)	negated
ERROR#, PEREQ, BUSY# (coprocessor)	negated

You can also enable and disable signals with the Shell Signal command.

Viewing and Modifying Memory

You can view and edit memory from the Memory window and by entering Dump, Write, Fill, Copy, and Search Shell commands.

Because reading and writing memory takes a small amount of processor time, which can degrade your program execution, memory access is initially disabled during emulation. Memory access is used in managing the Memory and Peripheral window displays and in changing memory contents with Memory, Peripheral, and Shell window

commands. To enable memory to be accessible during emulation, do one of the following before starting emulation:

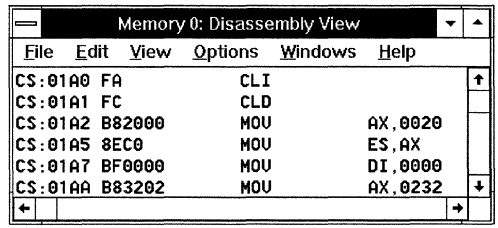
- Open the Toolbar Configure menu and enable Run Access.
- Enter RunAccess On on the Shell command line.

Changing the Memory Window Display

You can view memory as disassembly or numeric values in up to 20 independent Memory windows. Choose the desired format from each Memory window View menu. Multiple Memory windows are distinguished by a number from 0 through 19 in the title bar.

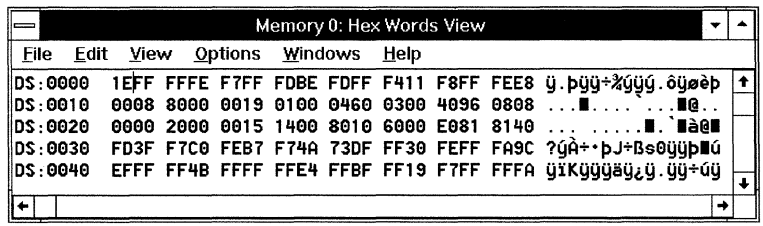
In the disassembly view, you can specify whether program symbols or their numeric addresses appear. Check (enable) or uncheck the Toolbar Configure menu Symbolic Disassembly item.

First-opened Memory window showing disassembly



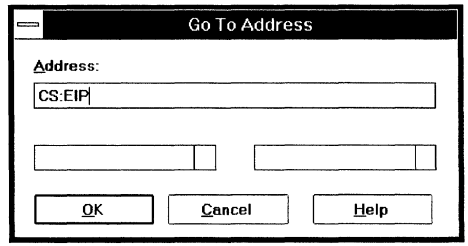
In a numeric view, memory appears as hexadecimal or decimal bytes, words, or double words followed by the ASCII equivalent, with dots representing non-printable characters.

First-opened Memory window showing hexadecimal words



To view another area of memory, double-click in the address column of the Memory window; or enter a numeric or symbolic address in the Edit menu Go To Address dialog box. A symbol must have a fixed address; that is, it cannot be a local variable or stack-resident parameter. Space and address mode options are greyed-out when unavailable.

Go To Address dialog box, accessed from the Memory window Edit menu Go To Address item, for displaying the current execution point in the Memory window



If you are unsure of a symbol name or an address, you can research it from the Shell command line:

DisplaySymbols	lists module, variable, and function names with line number and address information.
AddressOf	lists the address of a specified symbol.
NameOf	lists the symbol closest to a specified address.

To speed-up scrolling in the Memory window, choose the Options menu Read Ahead item. Using read-ahead near a non-existent memory region can cause a memory access failure.

Changing the Memory Contents

To change the memory contents, you can:

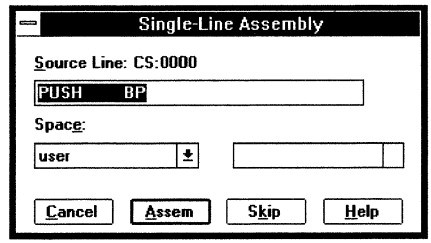
- Edit the hexadecimal, decimal, or ASCII values in the Memory window. Position the Memory cursor and overtype the display.
- Assemble code and data into memory using the Memory window Single-line Assemblerdialog box as described below.
- On the Shell command line, enter AsmAddr and Asm commands or Write, Fill, or Copy commands

To assemble lines of code into memory via the Memory window:

1. Check (enable) the Memory window View menu Disassembly item.
2. On the line to be changed, double-click anywhere except in the address column. The Single-line Assembler dialog box Source Line field shows the address and value of the line to be changed. (To close the dialog box without assembling, choose Cancel. Once a line is assembled, the Cancel button changes to a Close button.)
3. Type a line of assembly code in the dialog box.
4. Select the space and the operand/address size, as needed.
5. Choose Assem to write the code to memory and update the Memory window. The single-line assembler checks the syntax and reports any error without writing the erroneous line.

6. Repeat steps 3 through 5 to assemble subsequent lines. Choose Skip to leave a line unchanged.
7. Choose Close to close the dialog box.

Single-Line Assembly dialog box, accessed by double-clicking on a line of disassembly in a Memory window

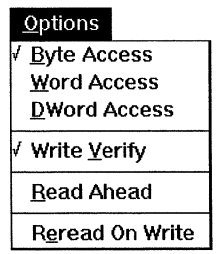


When the Memory window shows any view other than disassembly, you can edit the numeric and ASCII values. Position the cursor on the first value you want to change and type the new value. A value must fall within the range of the displayed radix. For example, in decimal byte radix the maximum value in a field is 255; if you try to replace 199 with 299, it is truncated to 200. An illegal (non-decimal or non-hexadecimal) entry causes a beep:

When you refresh the SLD window displays, changes to memory are reflected in all Memory windows, in the Source window disassembly, and in the Variable window values.

The numeric format displayed in the Memory window does not affect how memory is accessed. Memory access is set by the Size command or the Options menu, not by the View menu. For example, if Size=byte, memory accesses are byte-sized even when the Memory window display is Hex Words.

Memory window Options menu



Viewing and Modifying the Internal Peripheral Registers

This section applies to the PP-386EX, EA-386EX, SW-386EX, and EA-NS486 emulators.

Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Such access

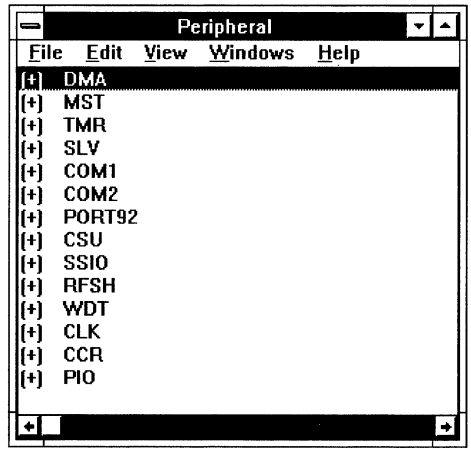
includes scrolling and refreshing the Memory and Peripheral windows and reading and writing memory from the Memory, Peripheral, and Shell windows. You can enable memory to be accessible during emulation; however, any such access can degrade your program execution. Before starting emulation, either:

- Open the Toolbar Configure menu and enable Run Access.
- On the Shell command line, enter RunAccess ON.

Changing the Peripheral Window Display

Registers are displayed hierarchically. At the top level are the peripheral mnemonics; then the registers for each peripheral; then the bit fields for each register. You can expand or compress each level. When the display is fully compressed, only the peripheral mnemonics appear.

Peripheral window for the SW-386 EX, showing the peripheral mnemonics fully compressed



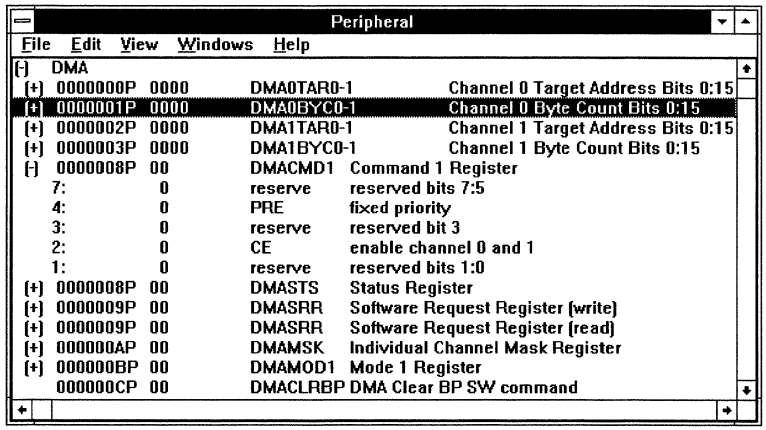
Expand a peripheral by clicking on the (+). The (+) changes to a (-) indicating the peripheral is expanded; a list of the peripheral's registers appears. Registers marked with (+) can be further expanded; click on the (+) to show the bit fields. Click on the (-) to recompress a line.

The register and bit field display columns are:

- The (+) or (-) expansion/compression indicator
- The register address; or, for a bit field, the bit number
- The field value
- The register or field mnemonic
- A description of the register or field

To display all peripherals and registers in expanded format, open the View menu and choose Expand All.

Peripheral window for the SW-386 EX showing the DMA peripheral expanded to registers and the DMACMD1 register expanded to bit fields



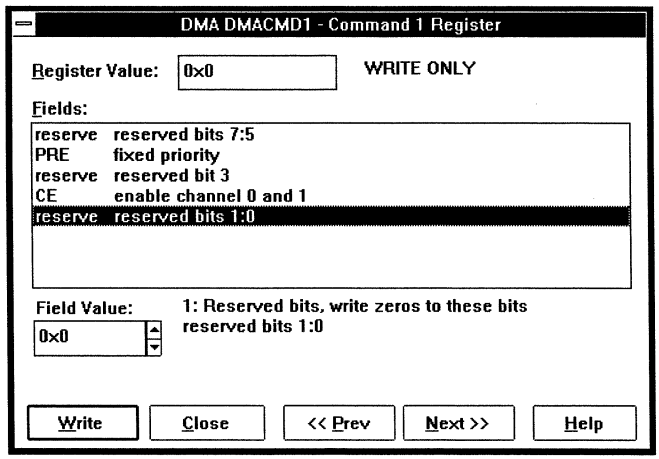
To navigate in the Peripheral window, enter a peripheral or register name or address in an Edit menu Go To... dialog box.

You can view the internal registers for each peripheral from the Shell command line with a Dump command. (For some 386EX and NS486 registers, use the IO argument.) Your processor may require setup before some peripheral registers are accessible. See your processor documentation.

Changing the Peripheral Register Values

Double-click anywhere on a register line; or select the register, open the Edit menu, and choose Register. You can edit the register and individual field values in the Register Edit dialog box.

Register Edit dialog box, accessed from the SW-386 EX Peripheral window display or Edit menu Register item.



You can modify the internal registers for each peripheral from the Peripheral window or from the Shell command line with a Fill, Copy, or Write command. (For some 386EX and NS486 registers, use the IO argument with these commands.) Your processor may require setup before some peripheral registers are accessible. See your processor documentation.

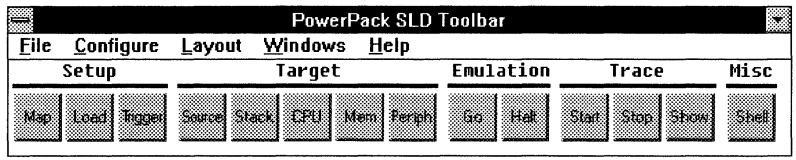
Debugging With Triggers and Trace

The PowerPack PP, EA, and SW emulators all collect trace during emulation. They differ in the amount of information collected and in the level of control you have over the trace collection. Triggers are available in the PP and EA emulators for complex control of emulation and trace collection. Events, also available in the PP and EA emulators, describe patterns of signal, data, and address bus activity for trigger conditions and trace search parameters.

Controlling Trace Collection

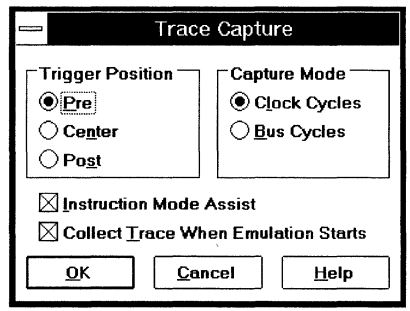
The SW and PP emulators start collecting trace when you start emulation. You can turn trace off and on during emulation with the Toolbar Trace Stop and Start buttons.

Toolbar showing the Trace Start, Stop, and Show buttons



In the EA emulator, you can specify whether trace collection starts with emulation. From the Trigger window Options menu or the Trace window Trace menu, choose Trace Capture and enable (check) or disable (uncheck) Collect Trace When Emulation Starts.

Trace Capture dialog box specifying trace collection to start when emulation starts

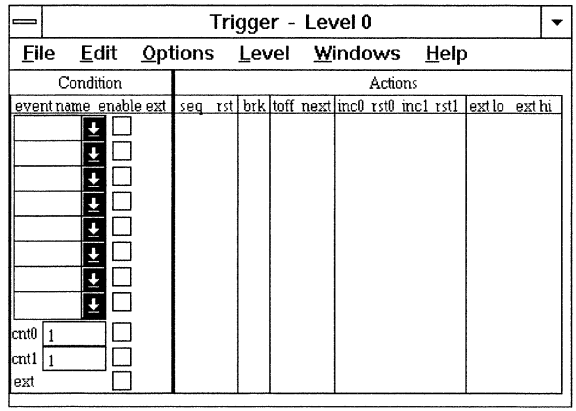


Automating Trace Capture

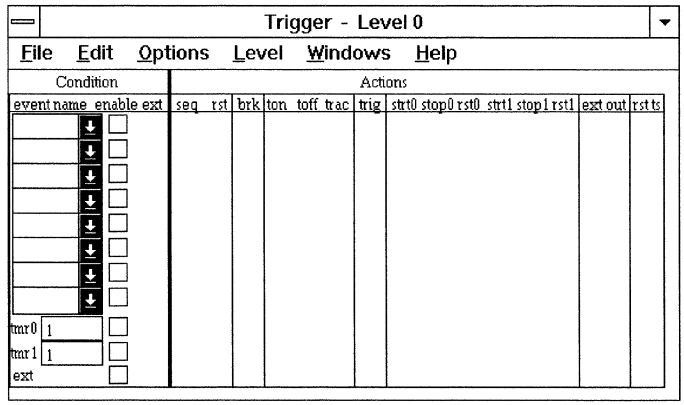
You can program the EA and PP emulators to automatically start and stop trace collection during emulation according to specified patterns of bus activity (called events) and other conditions. Such conditions with

their resulting actions are called triggers and are defined in the Trigger window.

PP emulator Trigger window, showing Toff and Next actions



EA emulator Trigger window, showing Ton, Toff, Trac, and Trig actions



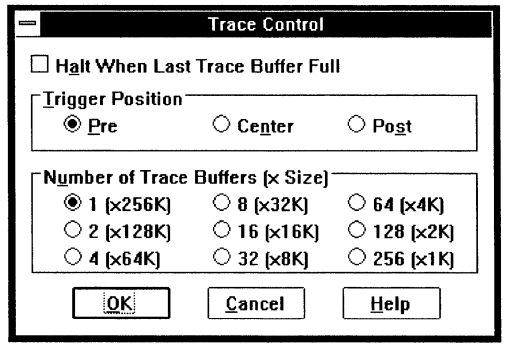
Trigger actions affecting trace include:

- stopping trace permanently a specified number of frames after the condition occurs (Toff in the PP; Trig in the EA)
- in the PP, with multiple trace buffers selected, closing the current buffer a specified number of frames after the condition occurs and starting subsequent trace in the next buffer (Next)
- in the EA, suspending trace immediately (Toff)
- in the EA, starting trace when the condition occurs (Ton)
- in the EA, collecting a single trace frame when the condition occurs (Trac)

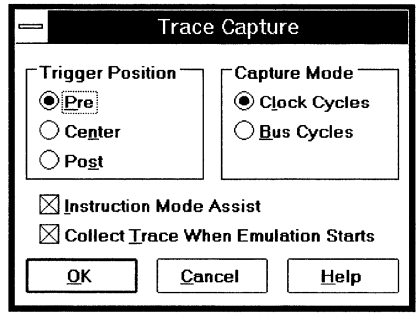
Each trace frame is a snapshot of the processor bus activity and other signals occurring during a single clock or bus cycle.

To specify the PP multiple buffers and the number of frames for delayed triggering in either emulator, use the PP Trace Control dialog box or the EA Trace Capture dialog box.

PP Trace Control dialog box, accessed from the Trigger window Options menu or the Trace window Trace menu



EA Trace Capture dialog box, accessed from the Trigger window Options menu or the Trace window Trace menu



In the PP, you can divide 256K frames of trace information among several buffers. Trace captured into each buffer is contiguous; trace in different buffers can come from separate parts of your program. For example, you can capture 256 separate blocks of 1K frames of trace , a single block of 256K trace frames, or various intermediate combinations. Each buffer is identified by a number, starting with 0, displayed in the Trace window title bar.

PP tracing starts in Buffer 0 when emulation starts and wraps around to overwrite the current buffer each time it fills up. The Next trigger action finishes filling the current buffer then starts filling the next buffer. If you enable (check) the Halt When Last Buffer Full box, tracing stops when all buffers have been filled. This operation overwrites the first buffer with several cycles after the end of the last buffer.

In the EA, you can delay the start of tracing until a trigger condition is met during emulation. To start emulation without tracing, disable (uncheck) the Trace Capture dialog box Collect Trace When Emulation Starts. Trigger actions to start tracing include:

Ton starts tracing with the frame in which the trigger occurs.

Trac captures only the frame in which the trigger occurs.

In the PP, the Toff action fills the current buffer then stops recording trace. In the EA, only one buffer is available. The Toff action stops recording trace immediately after the trigger frame. The Trig action fills the buffer then stops recording trace.

If multiple conditions are satisfied simultaneously, the emulator attempts to perform all the associated actions. For PP tracing, a Toff triggered by one condition can override a Next triggered by another condition. For EA tracing, simultaneous Ton, Toff, and Trac actions have no effect if tracing is previously on; or simulate a single Trac action if tracing is previously off.

For the EA Trig and the PP Toff and Next actions, you can specify approximately how many trace frames are saved after the frame in which the trigger occurs. In the Trace Control or Trace Capture dialog box Trigger Position field, enable:

Pre to collect no frames after the trigger. The trigger frame appears at or near the end of the buffer. In the PP, a few, frames can appear in the buffer after the trigger frame. In the EA, no frames are collected after the trigger frame.

Center to fill the buffer with an approximately equal number of frames before and after the trigger. The trigger frame appears in the middle of the buffer. In the EA, frames are collected for 125000 clock cycles following the trigger.

Post to fill the buffer with frames mostly after the trigger. The trigger frame appears at or near the beginning of the buffer. In the EA, frames are collected for 250000 clock cycles following the trigger.

After a Trig or PP Toff, trace is suspended until emulation halts and is restarted or until you manually start trace with the Toolbar Trace Start button or the Trace window Trace menu Start item. Both restarting emulation and manually starting trace clear previously collected trace.

You can collect all or a subset of the frames occurring after a Center or Post Trig action. To collect a block of frames within the Trig timer 125K or 250K clock cycle limit, define Toff and Ton triggers. To collect selected frames, define Trac triggers. The zero frame is the trigger frame; if trace is off when the Trig occurs, the zero frame is the next frame collected.

Formatting Trace Capture

The trace information varies between emulators, including the following for each bus cycle (SW emulators) or for each bus or clock cycle (PP and EA emulators):

- the frame number relative to either the triggering event or the instruction where tracing was stopped
- a timestamp for EA and PP emulators
- address bus values
- data bus values
- signal values, as listed in the *Hardware Reference*
- disassembled instructions

In the EA, you can capture more bus cycles or more detailed information by specifying trace frames to be bus-cycle or clock-cycle captures, respectively. In the Trace Capture dialog box Capture Mode field, enable:

Clock Cycles — to capture bus activity and other signals every clock cycle. Trace captured in this mode includes program activity and other processor messages and can be displayed as clock cycles, bus cycles, or disassembled instructions. The frame numbers are continuous.

Bus Cycles — to capture bus activity every bus cycle. The full set of bus pins is recorded as a unit, spanning multiple clock cycles if necessary, and can be displayed only as bus cycles. This capture mode covers more of your program execution but includes only the signals corresponding to program activity. For example, the branch messages required for disassembling trace are not captured.

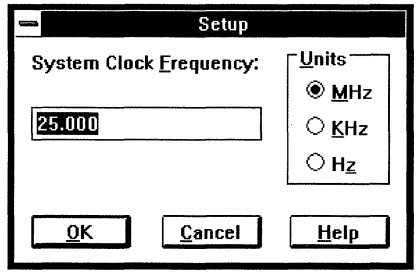
The PP captures only clock cycles, which can be displayed as clock or bus cycles or disassembled.

To be able to disassemble the trace, use clock-cycle capture and include information about branches taken. Include such information by enabling the PP Trace window View menu BTM Cycles item or the EA Trace Capture dialog box Instruction Mode Assist.

In both the PP and EA (regardless of the EA capture mode), you can specify the trigger to match conditions for either bus or clock cycles. For example, to match a condition that occurs during a single bus cycle but not within a single clock cycle, enable (check) the Trigger window Options menu Bus item.

The PP and EA timestamps count differently. The PP timestamp is computed from the target processor clock. Specify the clock speed in the System Clock Frequency Setup dialog box.

PP System Clock Frequency Setup dialog box, accessed from the Trace window Timestamp menu



The EA timestamp increments at 25 MHz, regardless of the target clock speed, with a range of approximately 733 minutes. For faster target systems, sequential frames can have identical timestamps.

To show elapsed time in trace, you can format the PP or EA timestamp relative to specific trace frames and reset the EA timestamp to 0 in various ways:

- Reset the EA timestamp any time, regardless of concurrent emulation or trace activity, by choosing the Trace window Timestamp menu Reset Timestamp Now item.
- Start the EA timestamp at 0 each time you start emulation, by initially enabling the Trace window Timestamp menu Reset Timestamp When Halted item.
- Reset the EA timestamp according to trigger conditions with the Rst Ts action.
- Show the time before and after a zero frame with the Trace window Timestamp menu Relative To Frame item.
- Show the incremental time between each frame with the Trace window Timestamp menu Delta item.
- Show the time since the EA timestamp was last reset with the Trace window Timestamp menu Absolute item.

To specify a zero frame for the relative timestamp, enter a frame number in the Trace window Timestamp menu Zero At Frame dialog box. The default zero frame is one of:

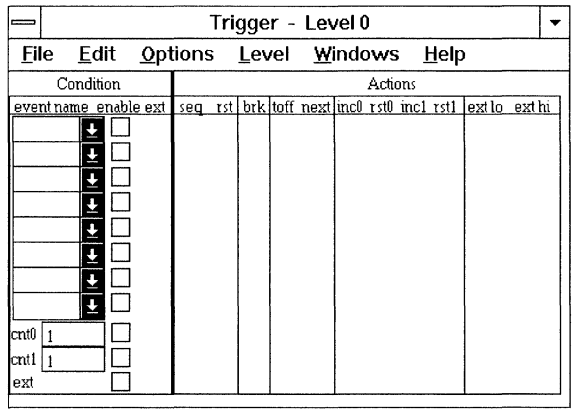
- the trigger frame of an EA Trig or PP Toff or Next action
- the last frame collected when trace stops for any other reason

Specifying Trigger Conditions

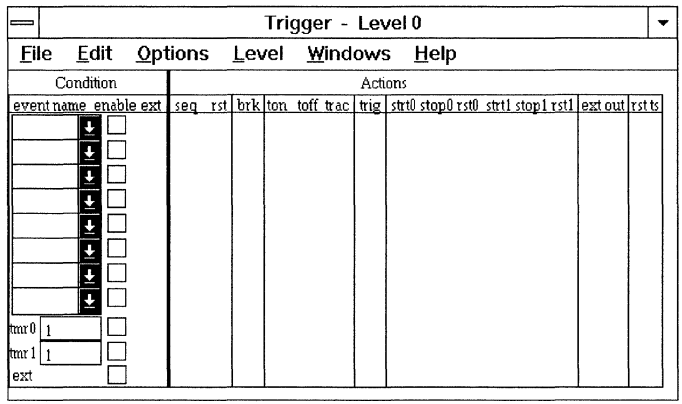
Trigger conditions are combinations of the following:

- A set of address bus, data bus, and signal values, called an event. The available signals differ between the EA and PP emulators and between the different processors supported for each emulator. See the *Hardware Reference*.
- The Trigger In pin on the front panel of the emulator. (See the *Hardware Reference* for an illustration of the Trigger In pin.)
- Counter and timer values based on clock cycles and event detection. The counter and timer options differ between the EA and PP.

PP emulator Trigger window with the counter option selected



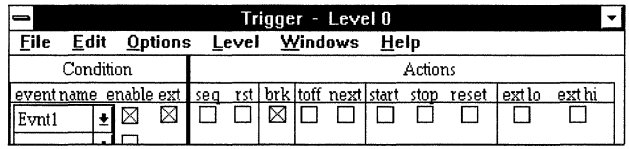
EA emulator Trigger window with one of the timer options selected



To use an event as a trigger condition, define one or more events; then, select an event name from an Event Name drop-down box in the Trigger window. Make the event an active condition by checking the Enable box next to the Event Name box.

To use the Trigger In signal as a separate event, check the enable box next to the Ext event below the event name boxes and counter or timer fields. To AND the Trigger In signal with an event, enable the event then check the Ext box in the same row as the event. In the PP, Trigger In is active-low. In the EA, you can specify Trigger In as active-high or active-low by selecting the appropriate Trigger window Options menu Trigger In Active item.

PP emulator Trigger window condition ANDing Evnt1 with the Trigger In (ext) signal



Conditions can be enabled simultaneously (on a single trigger level) or sequentially (on different trigger levels). The current trigger level, 0 through 3, appears in the Trigger window title bar. Display each level from the Trigger window Level menu.

To trigger on any of two or more conditions regardless of the order in which they occur, enable the conditions simultaneously. Each time one of the conditions occurs, the associated trigger actions are taken. If two or more conditions occur together, the emulator does all the associated actions. Some actions override others when done simultaneously:

- Toff (PP) overrides Next, turning trace off without starting another trace buffer.
- Ext Lo (PP) overrides Ext Hi, generating a low PP Trigger Out signal and no high Trigger Out signal.
- Rst0/1 overrides Inc0/1, resetting and not incrementing a counter.
- Stop (or EA Stop0/1) overrides Start (or EA Strt0/1), stopping a timer.
- Rst overrides Seq, activating the Level 0 trigger and not incrementing the trigger level.

Chaining Trigger Conditions

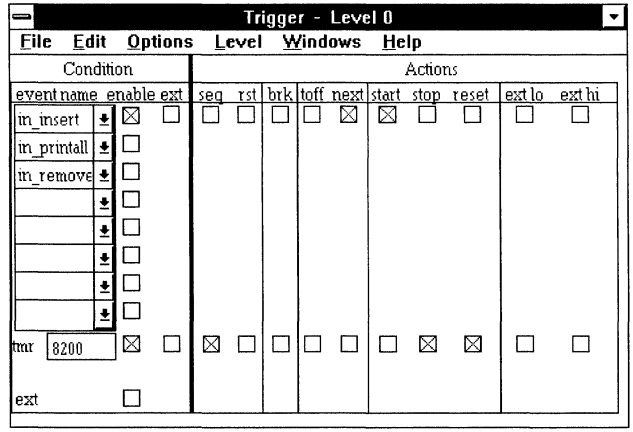
To avoid triggering on one condition until a prior condition has been met, enable the conditions on sequential levels. On the first level, enable the first condition and specify a Seq action. Seq suspends the current-level trigger and activates the next-level trigger. Disable the second condition on the first level and enable it on the next level.

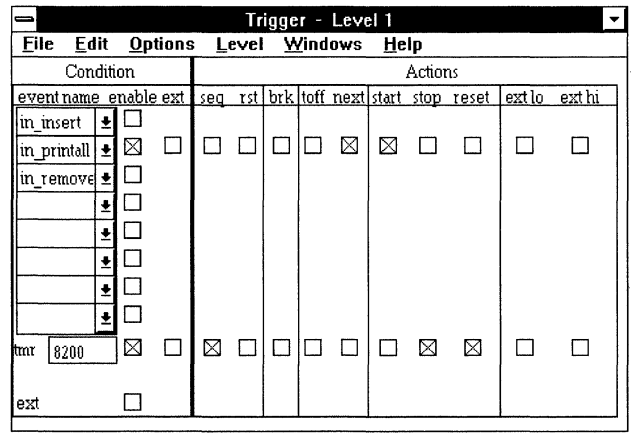
All levels must list the same set of up to eight events in the Event Name column and must specify the same counter or timer values; but the events and counters or timers can be enabled differently, can have

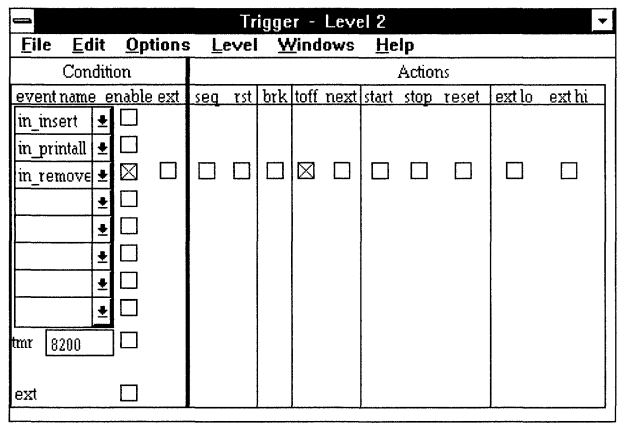
different external-trigger co-conditions, and can cause different actions at each level.

PP emulator Trigger sequential windows with:

- the in_insert event and the timer enabled at level 0
- the in_printall event and the timer enabled at level 1 (activated when in_insert occurs)
- the in_remove event enabled at level 2 (activated when in_printall occurs after in_insert has occurred)







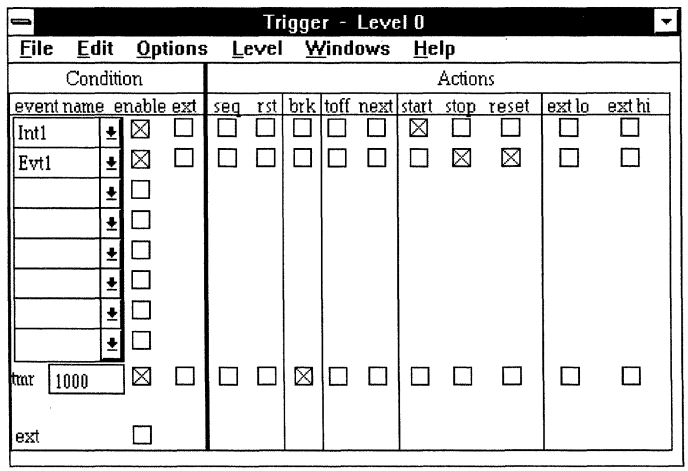
You can also schedule conditions by counting events or clock cycles. Select the paired counters, the single timer, or the EA paired timers from:

- the PP Trigger window Options menu Counter or Timer items
- the EA Trigger window Options menu 2 Counters, 2 Timers, or Cascaded Timer items

Enable an event to start the timer or counter and, optionally, events to stop and reset the timer or counter. Enable and fill-in the box beside the timer or counter as follows:

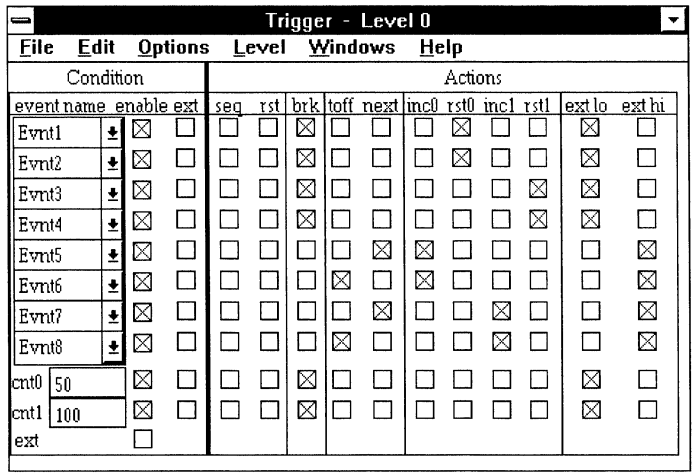
- A timer starts at 0 to count clock cycles when a start (for the tmr timer), strt0 (for the EA tmr0 timer), or strt1 (for the EA tmr1) action occurs. The timer increments at the clock rate of the emulation processor and wraps to 0 after reaching its maximum value. When the number of clock cycles specified in a timer box has elapsed, the timer action occurs. To stop a timer without resetting it, enable an event to do a stop, stop0, or stop1 action. Another event can restart the timer to continue. To reset a timer to 0 without stopping it, enable an event to do a reset, rst0, or rst1 action. In PP emulators, reset and stop a timer on a single condition by specifying both actions. In EA emulators, define two identical conditions, one with a reset action and the other with a stop action.

PP emulator Trigger window timer set to count 1000 clock cycles



- A counter starts at 0 and increments each time an inc0 (for cnt0) or inc1 (for cnt1) action occurs. When a counter reaches the number specified in its box, the counter action occurs. To reset a counter to 0, enable an event to do a rst0 or rst1 action.

PP emulator Trigger window counters set to count 50 instances of Evnt5 or Evnt6 between instances of Evnt1 or Evnt2 and to count 100 instances of Evnt7 or Evnt8 between instances of Evnt3 or Evnt4



Chaining Emulators

You can signal an external device with the PP ext lo and ext hi trigger actions or the EA ext trigger action. For example, when using multiple emulators in a multiprocessing target, an ext action from one emulator can appear as an ext condition in another emulator. The ext output signal appears on the Trigger Out pin on the emulator chassis, as described in the *Hardware Reference*.

In the PP, specify a low or high output by enabling the ext lo or ext hi action, respectively. In the EA, enable the ext action and choose the Trigger window Options menu Trigger Out Active High, Low, or Open Collector configuration.

Defining Events

An event is a combination of bus values:

Address — Reading or writing to a specific address, set of addresses, inside an address range, or "not" the described addresses. You can specify symbolic or numeric addresses.

Data — Reading or writing a specific value, set of values, range of values, or "not" the described values. You can specify symbolic or numeric data.

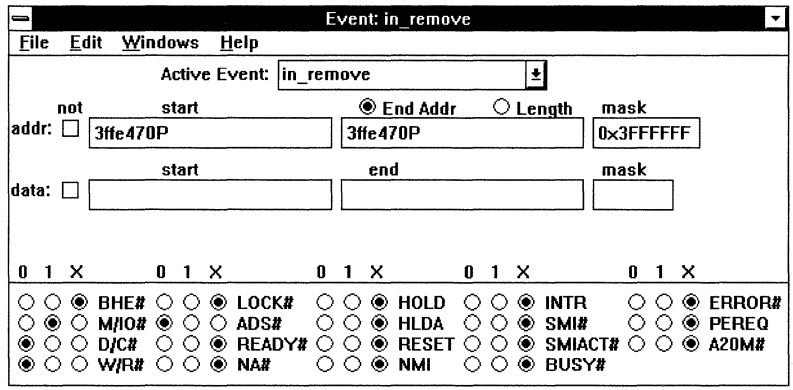
Signal — High or low logic levels on various processor signals. You can also specify don't-care for signals. For a list of supported signals, see the *Hardware Reference*.

Define an event in the Event window. Editing the Event window differs from editing a dialog box. The <Enter> key has no effect on the

field that you are editing. To ensure a field accepts an entry, move the cursor by clicking on another field or button. Pressing the <Delete> key to delete a highlighted value has no effect; press the space-bar instead.

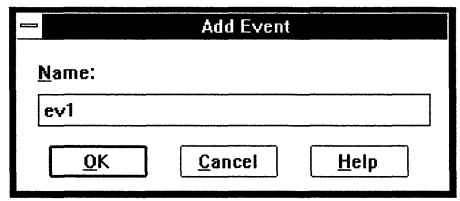
You can open the Event window from the Trigger or Trace window, by opening the Edit menu and choosing Events.

Event window, accessed from the Trigger or Trace window Edit menu



If no events are defined, the Add Event dialog box appears. Otherwise, to add a new event, in the Event window open the Edit menu, choose Add Event, and enter the new Event name.

Add Event dialog box, accessed from the Event window Edit menu, for creating a new event name



To define the address of an event: (If you don't care what addresses are accessed, leave all the Addr fields blank.)

1. Enter a symbolic or hexadecimal numeric address in the Addr Start field. This is the first address in the region where the event can occur.
2. Select End Addr or Length. Enter either the last address in the memory region where the event can occur, or the length in bytes of the region. If you specify no end address or length, the event is defined for the start address only.

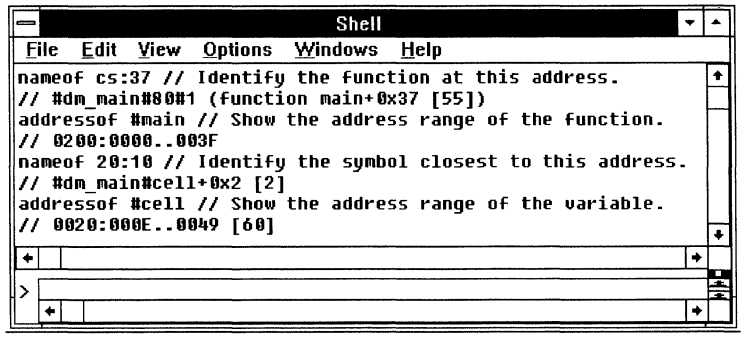
   If you are unsure of an address or address range, you can use the Shell window AddressOf and NameOf commands or the Source window Function pop-up menu.

Symbolic and numeric address-translation Shell

For example, use Shell commands as shown in the following to

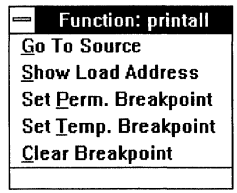
commands

find address information for defining an event relative to the dm_main module's main function or cell variable:

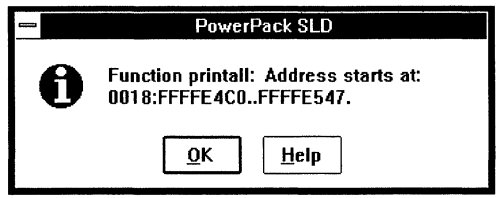


Another way to find the memory region of a function is via the Function pop-up menu. In the Source window, double-click on the function name and choose Show Load Address.

Function menu popped-up by double-clicking on the printall symbol in the Source window



Load address of the printall function



3. Optionally, you can enter a hexadecimal-AND mask value. The mask dictates which bits of the address are don't-care's (0) and which must match (1).
4. To match only addresses outside of the range or set you specified, check the Not box.

To define the data of an event: (If you don't care what data is read or written, leave all the Data fields blank.)

1. Enter numeric values in the Data Start and Data End fields. The emulator interprets the numbers as decimal unless you use the 0x prefix. For example, 10 is translated to 0x000A, and 0x10 is accepted as 0x0010.
2. Enter a hexadecimal-AND mask, using F's to match corresponding positions in the data pattern.

3. To match data outside of the specified range or set, check Not.

Specify signal states for the event by toggling the low (0), high (1) or don't care (X) buttons next to each signal mnemonic. Active-low signals are shown with a hash mark (#).

The signals available depend on the target processor, as described in the *Hardware Reference*. The mnemonic identifying each signal corresponds to the signal's primary function, regardless of whether you reconfigure the signal for other use.

You can define events in one emulator session and save them for reuse in another session. To save events to a file, in the Event window open the File menu and choose Save Events As. To retrieve saved events, choose Restore Events. Or, enter EventSave and EventRestore commands on the Shell command line.

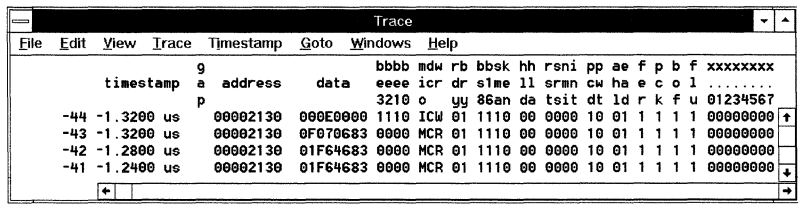
Viewing the Collected Trace

To display a trace buffer, open the Trace window. Only PP emulators support multiple trace buffers, navigable with the Goto menu Previous Buffer, Next Buffer, and Buffer items.

The Status window or icon message shows whether the emulator is tracing. You need not halt emulation to examine a snapshot of the collected trace. Each time trace stops, the Trace window is updated.

Read the abbreviated signal mnemonics vertically. For a list of supported signals, see the *Hardware Reference*.

Trace window with no trace collected



From the View menu, you can display trace as:

Clock mode	address, data, and signal values at each clock cycle (PP and EA emulators)
Bus mode	address, data, and signal values at each bus cycle
Instruction mode	disassembly of executed instructions and the memory accesses associated with the executed instructions

For the emulator to disassemble the trace information, you must have captured the clock-cycle branch-taken messages.

When viewing trace as disassembly (Instruction mode), you can link the Source and Trace window displays. When the windows are linked, scrolling the disassembled trace scrolls the corresponding source code synchronously. To link the Source and Trace cursors, do the following sequence

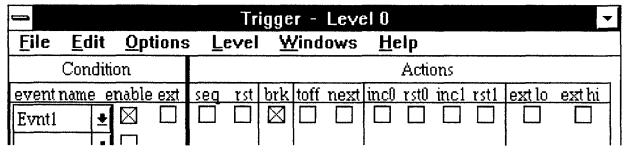
1. Choose the Trace window View menu Instruction item.
2. Enable the Trace window View menu Linked Cursor item.

Examples of Triggering

The illustrations in this section show PP 386CX emulator displays. The displays and options vary for different emulators and processors. This section demonstrates various trigger window configurations and describes their effects on emulation control.

Break Emulation

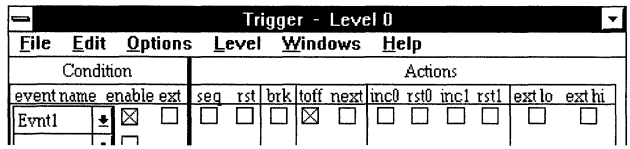
If Evnt1 occurs, emulation breaks.



1. Enable Evnt1 and choose the brk action.
2. Start emulation.
3. Tracing starts.
4. Emulation stops when the trigger occurs. (Tracing stops when emulation stops.)

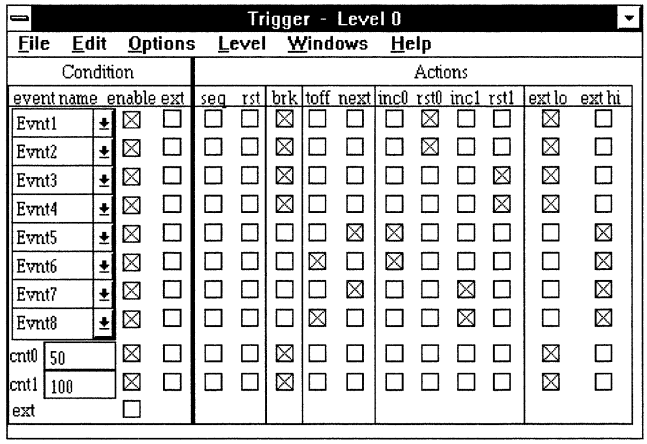
Stop Trace Without Breaking Emulation

If Evnt1 occurs, trace collection stops.



1. Enable Evnt1 and choose the toff action.
2. Start emulation.
3. When the trigger occurs, the trace buffer fills according to Trace Control; tracing stops; emulation continues.

Enable up to eight global events. Enabled events are processed in parallel. For this example, multiple trace buffers must be defined in the Options menu Trace Control dialog box and Counters must be selected in the Options menu.

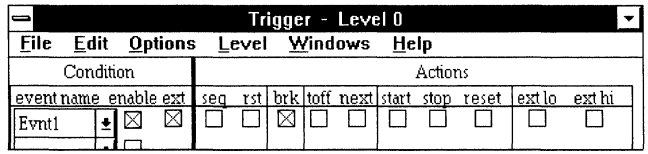


1. Enable the Event names in the eight drop-down list boxes.
2. Specify the actions to be taken when each event occurs:
   - Evnt1, Evnt2, Evnt3, and Evnt4 break emulation, reset one of the counters, and write 0 to the external trigger-out signal.
   - Evnt5 and Evnt7 fill the current trace buffer according to Trace Control and start collecting trace into the next trace buffer; increment one of the counters; and write 1 to the external trigger-out signal.
   - Evnt6 and Evnt8 stop tracing, increment one of the counters, and write 1 to the external trigger-out signal.
   - If Evnt5 and Evnt6 together occur 50 times without Evnt1 or Evnt2 occurring, cnt0 reaches 50, breaks emulation, and writes 0 to the external trigger-out signal.
   - If Evnt7 and Evnt8 together occur 100 times without Evnt3 or Evnt4 occurring, cnt1 reaches 100, breaks emulation, and writes 0 to the external trigger-out signal.

If multiple events occur simultaneously, all associated actions are taken. Some actions preclude others; for example, only ext lo or ext hi can occur when a brk also occurs.

AND an Event With an External Input

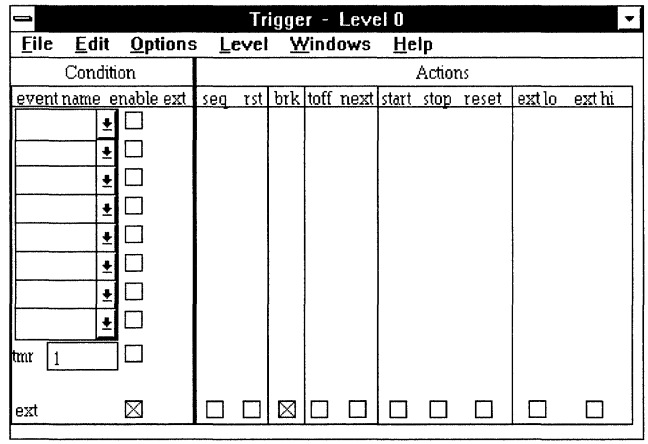
Logically AND the condition with an external trigger input low signal by checking the ext box (ext is to the right of enable).



The trigger condition is true when Evnt1 occurs during a low value on the emulator's external trigger input.

Trigger on External Input Alone

Enable ext on the last line of the Condition pane to set a trigger on an external signal alone (ext is located at the bottom of the left column).



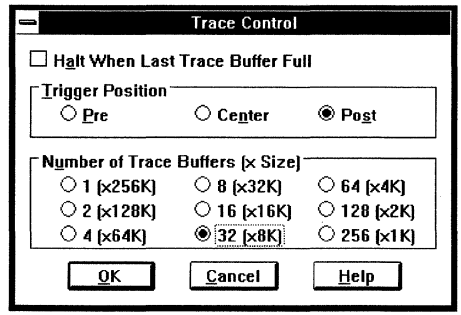
This condition is true when the emulator's external trigger input is low.

Define Sequential Triggers For Capturing Trace

Capture trace following each of three events in three separate trace buffers. This example uses a PP Intel386 CX emulator running the demo386.omf sample program installed with SLD.

Define buffers 8K bytes long. Position the trigger so the event appears near the beginning of the buffer (Post).

Trace Control dialog box specifying 32 buffers of 8K bytes each, with the trigger frames near the beginning of each buffer

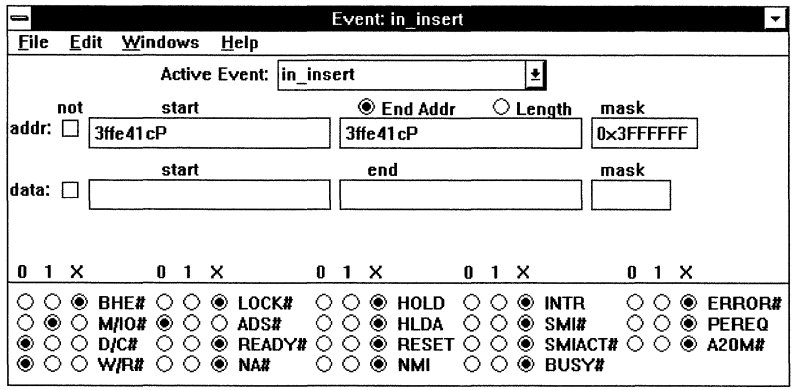


Define an event at the first code location inside each of three function calls: insert, printall, and remove. To find the addresses, use Xlt:

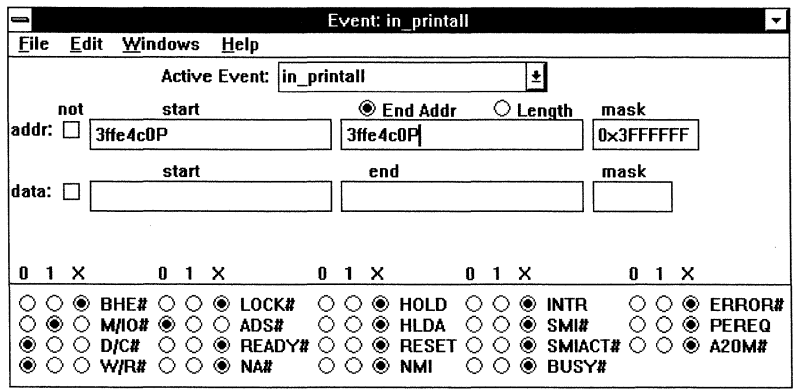
```
Xlt #insert;
// 0018:FFFFE41C = FFFFE41CL = 3FFE41CP
```

The following figure shows the three event definitions.

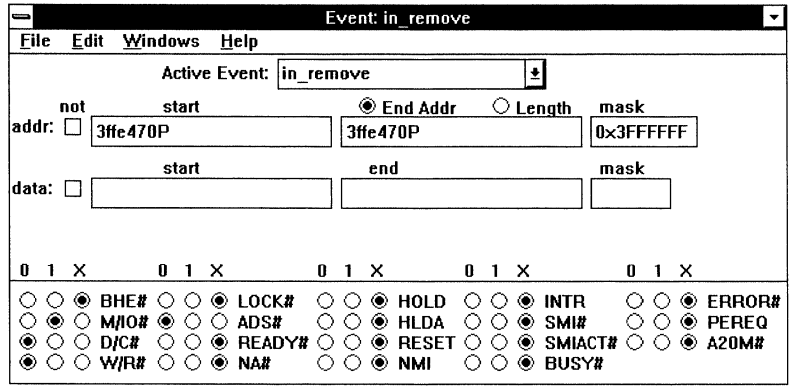
Event window defining a Memory Code Read event in the insert function



Event window defining a Memory Code Read event in the printall function



Event window defining a Memory Code Read event in the remove function

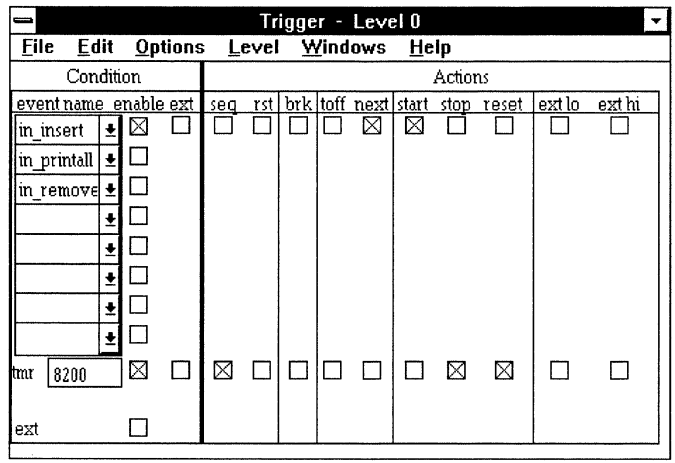


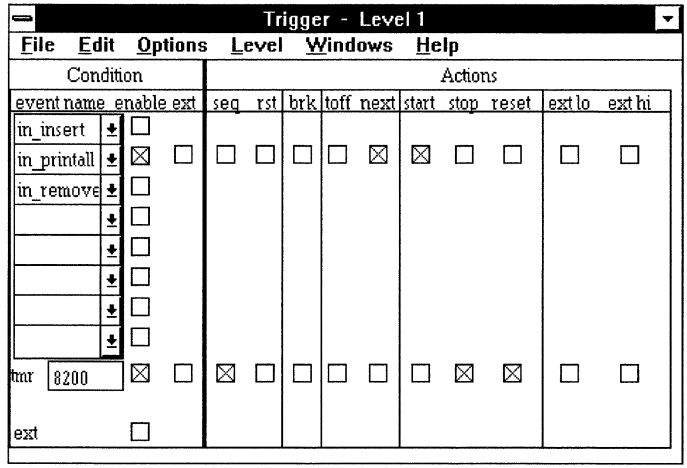
Enable the Options menu Clock, setting the event trigger to respond to clock cycles.

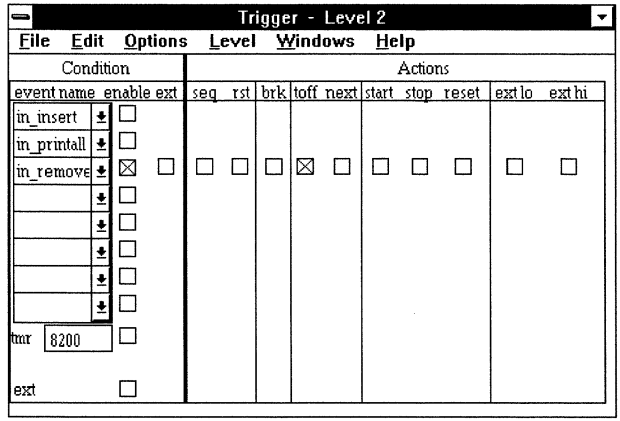
Enable the Options menu Timer, displaying a tmr line at the bottom of the Condition pane. Check the tmr enable box. Type 8200 in the tmr value field, specifying 8200 clock cycles to elapse between timer triggers. This demo program is so small that the events defined for the triggers occur multiple times in the trace captured to post-fill an 8K-byte trace buffer. Since only one trace-control action (toff, next) can occur in each buffer, the timer ensures that tracing moves to the next buffer before sequencing to the next trigger.

PP emulator Trigger sequential windows with:

- the in_insert event and the timer enabled at level 0
- the in_printall event and the timer enabled at level 1 (activated when in_insert occurs)
- the in_remove event enabled at level 2 (activated when in_printall occurs after in_insert has occurred)







Each of the first two triggers captures trace following its event and starts a timer to run while the buffer fills. When the buffer is full, tracing begins in the next buffer. When the timer finishes, it stops, resets itself, and arms (sequences to) the next trigger.

The final trigger turns trace off, filling the current buffer. Emulation continues but trace does not.

powerpak.ini File Reference

This chapter describes the contents of the powerpak.ini *file.*

The SLD software installation creates the powerpak.ini file in your Windows directory.

Backup CAUTION

Always back up powerpak.ini. *Once you have modified* powerpak.ini, *you may need to restore the default contents by reinstalling the SLD software.*

The following sections can appear in powerpak.ini:

Section	Purpose
[Comm]	Host-to-emulator communication
[CPUInfo]	Intel debug register allocation
[DefaultLayout]	Window screen locations
[InitScript]	Script file to run on invocation
[LoadOptions]	Load options
[Network]	Network information
[Serial]	Host system COM port number
[SourceInfo]	Source window Go, Step, and View options
[StackInfo]	Stack window options
[StatusInfo]	Status window options
[SystemInfo]	Target support
[ToolBarInfo]	Toolbar configuration options
[ToolChain]	Section names and bitfield information
[TraceInfo]	Trace Control and Trigger window options
[TrigInfo]	Trigger window options
[VariableInfo]	Toolchain Variable window options

Many entries are toggle settings with possible values of 1 or 0. For such entries, 1 is enable and 0 is disable.

Whenever possible, change entries using menus or Shell commands rather than modifying powerpak.ini in a text editor.

Avoid modifying any entry not documented in this chapter.

[Comm]

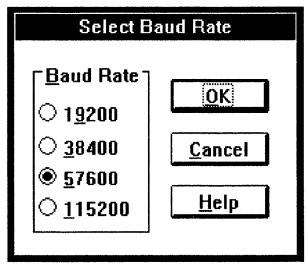
describes host/emulator communication

type=[serial | pcnfs | lanserver] describes how the emulator communicates with your host system. This entry is set to serial by the SLD installation and changed by the PP emulator network installation. If your network configuration changes, affecting communication between the host system and the PP emulator, edit powerpak.ini.

serial	specifies serial communication.
pcnfs	defines the emulator as a node on a PC-NFS network.
lanserver	defines the emulator as a node on an OS/2 LAN server.

BaudRate=[19200 | 38400 | 57600 | 115200] specifies the baud rate for communication between the emulator and your host system. The first time you start the SLD software, you must specify a baud rate. For some host systems, baud rates above 57600 require a special Windows driver.

Select Baud Rate dialog box, popped-up automatically the first time you start the SLD software



powerpak.ini lines specifying serial communication at 57600 baud

```
[Comm]
// This is the installed default communication type.
// Network installation changes this entry.
type=serial
// The BaudRate and com port (in the [Serial] section) are
// unspecified until you fill-in the appropriate dialog boxes.
BaudRate=57600
```

[CPUInfo]

allocates debug register use

dr [<num>]=[user | system] specifies whether each debug register is reserved for use by your program or by the emulator for hardware breakpoints.

<num>	specifies the debug register as 0, 1, 2, or 3.
user	enables your program's access to the debug register.
system	reserves the debug register for use by the emulator,

blocking your program's access to the register.

powerpak.ini lines added by a DR 1 USER Shell command

```
// The emulator adds this section when you use a DR command,
// as described in the "Shell Window Reference" chapter.
[CPUInfo]
dr 0=system
dr 1=user
dr 2=system
dr 3=system
```

[DefaultLayout]

specifies Window screen dimensions

The<SLDWindow>Presenter=[<Dimensions>] specifies the screen locations and sizes for the initially displayed SLD windows.

Move and resize the SLD windows using the Windows mouse or cursor. To save the layout without exiting the SLD software, choose the Toolbar Layout menu Save Layout Now item. If you are likely to change the layout again before exiting but want the same initial layout the next time you start, disable the Layout menu Save Layout On Exit item.

The emulator fills-in this section when you save the layout.

[InitScript]

defines which Shell script file executes when you invoke SLD

script=[<scriptFile>] sets <scriptFile> as the initialization script (the file of Shell commands run each time you start the SLD software). Either specify a full pathname or put the script in the SLD directory (e.g., c:/powerpak). When no <scriptFile> is specified, none runs.

To change this entry, edit powerpak.ini.

powerpak.ini lines specifying sample initialization script

```
[InitScript]
// The sample script include.me is installed with the SLD
software.
script=include.me
```

[LoadOptions]

specifies load options

[LoadOptions] entries can be changed in the Load Options dialog box. To open the Load Options dialog box, choose the Toolbar Load button; or choose the Source window File menu Load Code item. In the Load dialog box, after browsing the filename to be loaded, choose the

Options button. Load command arguments override the [LoadOptions] entries.

AddressSpace=[user | smm] specifies Intel x86 SMM or User address space when the file is loaded. Choose the Load Options dialog box User or SMM button.

LoadSymbol=[1 | 0] specifies whether symbols are loaded. For example, when symbols are already loaded, turn off symbol loading and load only code. Toggle the Load Options dialog box Load Symbols item.

LoadCode=[1 | 0] specifies whether to load code. For example, when debugging in ROM, turn off code loading and load only symbols. Toggle the Load Options dialog box Load Code item.

LoadReportStatus=[1 | 0] specifies whether the load progress indicator appears during loading. Toggle the Load Options dialog box Report Status item.

LoadReportWarning=[1 | 0] specifies whether load warnings appear. Toggle the Load Options dialog box Report Warnings item.

LoadOnDemand=[1 | 0] specifies whether symbolic information is loaded for all modules immediately or loaded only when needed. Symbolic information includes local symbol and line-number information for a module. Such information is needed when either the module is displayed in the Source window or a breakpoint is set in the module. Advantages of on-demand symbol loading include faster initial loading, faster lookup for the symbols that are demanded, and less memory occupied by the loaded file since only the required symbols are loaded. Toggle the Load Options dialog box On Demand Symbol Loading item.

LoadDemangle=[1 | 0] specifies whether symbols are demangled for the first instance of each overloaded function in a C++ program. Toggle the Load Options dialog box Demangle C++ Names item.

LoadUpdateBase=[1 | 0] specifies whether x86 symbol base addresses are updated. For example, if your descriptor table bases are nonzero, you can save time by having the load process update your symbol base addresses from the descriptor table information. Toggle the Load Options dialog box Update Symbol Bases item. This option must be used in conjunction with LoadRegister (toggle the Load Options dialog box Load Initial Registers item).

LoadRegister=[1 | 0] specifies whether x86 initial register values are loaded. For example, if your initialization code does nothing but initialize the registers, you can save time by having the load process extract the register information from your initialization code. Then,

you need not execute the initialization code. Toggle the Load Options dialog box Load Initial Register Values item.

powerpak.ini lines to load all code and symbols immediately (into User space, if there is a choice) and to report progress while loading

```
[LoadOptions]
// 1=enable, 0 = disable
// The following are the installed default values.
LoadSymbol=1
LoadCode=1
LoadReportStatus=1
LoadReportWarning=0
LoadOnDemand=0
LoadDemangle=0
LoadUpdateBase=0
LoadRegister=0
// The following is installed for some processors.
AddressSpace=User
```

[Network]

lists available PP emulators

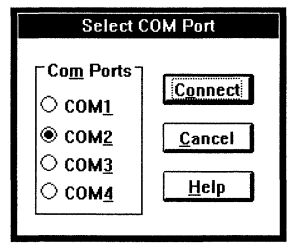
emulators=<name>[,<name>...] specifies one or more PP emulators installed on the network. When multiple <names> appear in the list, a dialog box appears so you can choose one. This section is added by the network installation. Change this entry by editing powerpak.ini directly.

[Serial]

defines the COM port attached to the emulator or debugger hardware

comport=com[1 | 2 | 3 | 4] sets the COM port connecting your host system with the emulator. The first time you start the SLD software, this dialog box adds the [Serial] section and comport entry. To change the COM port later, edit powerpak.ini.

Select COM Port dialog box, popped-up automatically the first time you start the SLD software



powerpak.ini lines to use COM port 2

```
[Serial]
comport=com2
```

[SourceInfo]

controls the Source window display and options

DisplayLineNum=[0 | 1] specifies whether source line numbers are displayed in the Source window. Toggle the Source window View menu Line Number item.

StepCount=<num> specifies how many steps (1 to 0x7FFFFFFF) are executed per Step command. Choose the Source window Options menu Step Count item and fill-in the dialog box. Or, enter a Step or StepSrc Shell command.

ViewSource=[1 | 0] specifies the Source window display either as source from the source file (1) or as a combination of source and disassembly (0). Choose the Source window View menu Source Only item or Mixed Source And Assembly item.

UseGoInto=[1 | 0] specifies whether the Source window Call and Return buttons perform Go Into (1) or Go Until (0) emulation. Open the Source window Options menu Set Go Buttons item; choose Until Call/Return or Into Call/Return.

UseLineExecGranularity=[1 | 0] specifies whether a step executes an entire source line (1) or a single source statement (0). Open the Source window Options menu Set Step Granularity item and choose Source Line or Source Statement. Or, enter a StepSrc Line or StepSrc Statement Shell command.

HistoryDepth=<num> specifies how many source browsing locations (1 to 50) are saved. Fill-in the Source window Options menu Browser History Depth dialog box.

TabWidth=<num> specifies the number of spaces (1 to 32) that replace a tab character in the Source display. The installed default is TabWidth=8. Fill-in the Source window Options menu Tab Width dialog box.

SourceDelimiterUseCRLF=[1 | 0] specifies the source delimiter (the ASCII character string used by the debugger to delimit a source line) as carriage return/linefeed (1), the DOS newline string or as linefeed only (0), the UNIX newline string. When SLD is installed, the delimiter is carriage return/linefeed. Open the Source window Options menu Source Line Delimiter item; choose Carriage Return/Linefeed or Linefeed Only.

OperandAddressSize=[0 | 1 | 2] specifies the x86 address mode for viewing disassembly in the Source window as:

0 derives the address mode based on the pmode.

1 uses 16-bit address mode.

2 uses 32-bit address mode.

Open the Source window View menu Operand/Address Size item; choose Auto, Use16, or Use32.

DefaultModuleExtensions=[C, ASM, CPP, CXX, S] specifies the default source file extensions. To change this entry, edit powerpak.ini. When the source filename is stripped of its extension, the emulator searches for the filename with the default module extension.

LoadFile[0 | 1 | 2 | 3]=<pathname> specifies the pathnames of the last four source files you have loaded. This entry is updated automatically when you load a module with associated source.

NumAliasPath=<number> specifies how many directories are listed as source paths. This entry is updated automatically when you add or delete a source path.

SourcePathAlias<num>=<path> specifies a source path. There are as many of these entries as are counted in NumAliasPath. A SourcePathAlias entry is added, changed, or deleted each time you add, change, or delete a source path. Choose the Source window Options menu Source Path item. In the Source Path dialog box, to add a new path, choose Add and fill-in the Add dialog box; to change a path, select the path, choose Edit, and fill-in the Edit dialog box; to delete an existing path, select the path and choose Delete.

powerpak.ini lines specifying Source window options for associating source files with modules, displaying source or disassembly, and stepping

```
[SourceInfo]
// The following are the installed default values.
// 1 = enable, 0 = disable
DisplayLineNum=1
StepCount=1
ViewSource=1
UseGoInto=1
UseLineExecGranularity=1
HistoryDepth=10
TabWidth=8
SourceDelimiterUseCRLF=1
// 0=auto, 1 = use16, 2 = use32
OperandAddressSize=0
// default source module extensions
DefaultModuleExtensions=C,ASM,CPP,CXX,S
LoadFile0=
LoadFile1=
LoadFile2=
LoadFile3=
// The following entries are not installed, but
// are added when you display source.
NumAliasPath=
SourcePathAlias0=
```

[StackInfo]

controls the display and other options in the Stack window.

StackSize=<num> specifies the stack size and must match the target's allocated stack size. Unless specified in the load file, the stack size defaults to 4K bytes. Fill-in the Stack window Options menu Stack Area dialog box; or in the Shell window enter a SetStackArea or SetStackSize command.

StackBaseAddr=<hex_addr> specifies the stack base address, as defined in the load file. Fill-in the Stack window Options menu Stack Area dialog box; or in the Shell window enter a SetStackArea or SetStackBase command.

PercentAlarmLimit=<num> specifies the alarm limit as a percentage of the stack size, from 1 to 100. Fill-in the Stack window Options menu Alarm Limit dialog box; or in the Shell window enter a SetStackAlarm command.

EnableAlarmLimit=[1 | 0] specifies whether the emulator displays a warning message when stack usage reaches the percentage of the stack area specified by PercentAlarmLimit. Toggle the Stack window Options menu Enable Alarm Limit item; or in the Shell window enter EnableAlarmLimit or DisableAlarmLimit.

EnableHWM=[1 | 0] enables or disables the high water mark. Toggle the Stack window Options menu Enable High-Water Mark item; or in the Shell window enter EnableHighWaterMark or DisableHighWaterMark.

ViewStackAddr=[1 | 0] enables or disables displaying the Stack window stack address (the location of the frame on the stack). Toggle the Stack window Options menu Include Stack Address item.

ViewCodeAddr=[1 | 0] enables or disables displaying the Stack window code address (the called function's return destination). Toggle the Stack window Options menu Include Code Address item.

powerpak.ini lines specifying options for stack usage statistics

```
[StackInfo]
// The following are the installed default values.
StackSize=1024
StackBaseAddr=0x800
PercentAlarmLimit=95
// 1 = enable, 0 = disable
EnableAlarmLimit=0
EnableHWM=0
ViewStackAddr=1
ViewCodeAddr=1
```

[StatusInfo]

specifies whether the Status window appears on top of other windows

Topmost=[1 | 0] specifies whether the Status window (or icon, when minimized) appears on top of other SLD windows. With Topmost = 1, the Status window or icon remains in the foreground relative to any other overlapping SLD window, regardless of which window is in focus. Toggle the Status window Control menu Always on Top item.

powerpak.ini lines positioning the Stack window

```
[StatusInfo]
// The following is the installed default value.
Topmost=1
```

[SystemInfo]

co-ordinates Intel386 emulation and target processors

386EmulatorCPU=[386CX A-step | 386CX B-step | none] describes the CX or SX bondout processor in the emulator probe.

386EXEmulatorCPU=[386EX A/B-step | 386EX C-step] describes the EX bondout processor in the emulator probe.

386TargetCPU=[386SX | 386CXSA | 386CXSA - 5V | 386CXSB | 386CXSB - 3V] describes the CX or SX processor in your target design.

386EXTargetCPU=[386EXTA | 386EXTB | 386EXTB - 3V | 386EXTC | 386EXTC - 5V] describes the EX processor in your target design.

386EmulatorCPUs=386CX A-step,386CX B-step lists the Intel386 CX/SX bondout processors recognized as emulator processors.

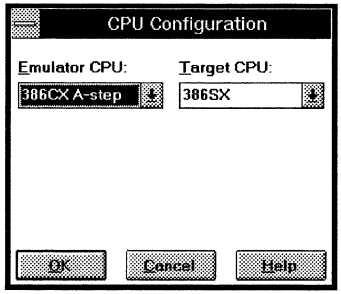
386EXEmulatorCPUs=386EX A/B-step,386EX C-step lists the Intel386 EX bondout processors recognized as emulator processors.

386TargetCPUs=386SX,386CXSA,386CXSA - 5V,386CXSB, 386CXSB - 3V lists the Intel386 CX/SX processors recognized as target processors.

386EXTargetCPUs=386EXTA,386EXTB,386EXTB - 3V, 386EXTC,386EXTC - 5V lists the Intel386 EX processors recognized as target processors.

The first time you start the SLD software for Intel386 emulation, a dialog box appears for the 386[EX]EmulatorCPU and 386[EX]TargetCPU entries. To change these entries later, edit or reinstall powerpak.ini.

CPU Configuration dialog box for co-ordinating the emulator's bondout processor with your target processor



To discover the stepping, look for the part number (FPO) on the chip. Production FPOs are 8 digits followed by a change indicator. Pre-production and obsolete parts use a 5-digit code starting with Q.

CPU	Step	Production FPO	Pre-production FPO
386EX	A	xA or xB	Q8492
	B	xD	Q7949
	C		Q8042
386CX or SX	A	xA	Q8307
	B	xB	Q8543

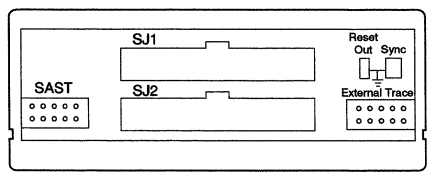
To discover the current settings, use the Version Shell command.

targResConfig=<Configuration> specifies the asserted and negated states of the SW and EA emulator Reset Out signal (Reset Target Shell command):

Configuration	Asserted	Negated
OpenCollector	low	high-Z
ActiveLow	low	high
ActiveHigh	high	low

To change entries in this section, edit powerpak.ini.

EA and SW emulator front panel, showing the Reset Out pins



powerpak.ini lines for co-ordinating the emulator and target processors

```
[SystemInfo]
// The emulator fills-in the following entries
// when you fill-in the appropriate dialog box.
386EmulatorCPU=
386TargetCPU=
386EXEmulatorCPU=
```

```
386EXTargetCPU=
// Avoid changing the following entries.
386EmulatorCPUs=386CX A-step,386CX B-step
386TargetCPUs=386SX,386CXSA,386CXSB
386EXEmulatorCPUs=386EX A/B-step, 386EX C-step
386EXTargetCPUs=386EXTA,386EXTB,386EXTB - 3V,386EXTC -
5V
// The emulator adds this entry the first time
// you enter a Reset Target command.
targResConfig=OpenCollector
```

[ToolBarInfo]

saves the window layout and masks interrupts during single stepping.

SaveLayoutOnExit=[1 | 0] specifies whether the SLD window layout (the SLD windows as you have opened, positioned, and sized them) is saved when you exit. If the layout is not saved, the previously saved or default layout appears next time you start. Toggle the Toolbar Layout menu Save Layout On Exit item.

powerpak.ini lines to retain the prior layout when exiting

```
[ToolBarInfo]
// The following is the installed default value.
SaveLayoutOnExit=0
```

[ToolChain]

describes OMF86 section names and bitfield information

OMFBaseTypeNames=CODE,DATA specifies your OMF86 code and data section names. Edit powerpak.ini to change this entry.

maxBitFieldSize=[16 | 32] specifies your OMF86 bitfield size. Use the MaxBitFieldSize Shell command to specify 16 for loadfiles generated with the Borland C compiler and 32 for other toolchains.

powerpak.ini lines resolving OMF86 toolchain specifics

```
[ToolChain]
// The following are the installed default values.
// OMF86 Base type names
OMFBaseTypeNames=CODE,DATA
// OMF386 - maxBitFieldSize [<16|32>]
maxBitFieldSize=32
```

[TraceInfo]

sets the Trace window options

linkedCursor=[on | off] turns on or off the code address link between the Trace and Source windows. The link is valid only when the Trace

window displays instructions (see viewType in this section) and the Source window displays mixed source and disassembly (see viewSource in the [SourceInfo] section).

When cursors are linked, the Source window scrolls automatically to match the Trace display.

To enable linkedCursor:

1. Enable the Source window View menu Mixed Source And Assembly item.
2. Enable the Trace window View menu Instruction Cycles item.
3. Enable the Trace window View menu Linked Cursor item.

To disable linkedCursor, disable the Trace window View menu Linked Cursor item.

viewType=[bus | clock | instruction] sets the trace view as:

bus	displays the processor signals at each bus cycle.
clock	displays the processor signals at each clock cycle (PP and EA emulators only).
instruction	displays the instructions executed by the processor and the resulting reads and writes.

Choose the Trace window View menu Clock, Bus, or Instruction Cycles item.

timestamp=[on | off] turns on or off the PP and EA trace timestamp display. Toggle the Trace window View menu Timestamp item.

systemFrequency=<frequency> specifies the PP emulator target system clock frequency; 0.01 Hz ≤ <frequency> ≤ 40 MHz. Fill-in the Trace window Timestamp menu Setup dialog box.

tsmode=[relative | delta | absolute] specifies the PP or EA timestamp mode as:

relative	shows timestamps as elapsed time from a zero frame.
delta	shows each timestamp as incremental time from the previous frame.
absolute	shows EA timestamps as elapsed time from the last timestamp reset.

Choose the Trace window Timestamp menu Relative To Frame, Delta, or Absolute item.

tsReset=[on | off] specifies whether the EA timestamp is set to 0 each time emulation halts.

captureMode=[clock | bus] specifies whether the EA captures trace

as clock or bus cycles.

traceStartState=[enabled | disabled] specifies whether the EA starts capturing trace when emulation starts.

btmCycles=[enabled | disabled] specifies whether BTM (branch-taken message) cycles are collected and shown. A BTM cycle indicates a change in execution flow, such as a jump. The emulator must collect BTM cycles to display trace as instructions. Toggle the PP Trace window View menu BTM Cycles item or the EA Trace Capture dialog box Instruction Mode Assist item.

powerpak.ini lines specifying Trace options

```
[TraceInfo]
// The following are the installed default values.
linkedCursor=off
viewType=bus
timestamp=on
systemFrequency=25MHz
tsMode=relative
tsReset=on
captureMode=clock
traceStartState=enabled
btmCycles=enabled
```

[TrigInfo]

sets the Trace Control and Trigger window options

This section is used by the EA and PP emulators only.

numTraceBuffers=[1 | 2 | 4 | 16 | 32 | 64 | 128 | 256] specifies the number of PP trace buffers. Specifying the number of trace buffers also specifies the size of each trace buffer. The buffer size options depend on the amount of trace memory (128K or 256K bytes) in your emulator.

Fill-in the PP Trace window Trace menu or Trigger window Options menu Trace Control dialog box Number Of Trace Buffers (X Size) item.

traceAlignment=[center | pre | post] specifies the position of the triggering event in the trace buffer:

center	Trace buffers fill before and after the trigger. The trigger appears in the center of the trace display.
pre	Trace buffers fill up to the trigger. The trigger appears near the end of the display.
post	Trace buffers fill up after the trigger. The trigger appears near the beginning of the display.

Fill-in the Trace or Trigger window Options menu PP Trace Control or EA Trace Capture dialog box.

breakOnFull=[on | off] specifies whether the emulator breaks when all PP trace buffers become full. Toggle the Trace window Trace menu or the Trigger window Options menu Trace Control dialog box Halt When Last Trace Buffer Full item.

counterTimer=[counter | timer | timerx2] configures the Trigger window counter and timer conditions:

counter	enables the PP single counter or the EA paired counters. Choose the Trigger window Options menu PP Counter or EA 2 Counters item.
timer	enables the paired timers. Choose the Trigger window Options menu PP Timer or EA 2 Timers item.
timerx2	enables the EA single timer. Choose the EA Trigger window Options menu Cascaded Timer item.

trigMode=[bus | clock] specifies the type of cycle used for triggering:

bus	automatically samples processor pins at the proper time in a bus cycle. The trigger is based on aligned samples.
clock	triggers on any cycle coming from the processor, regardless of whether it is a valid bus cycle. Use clock triggering to trigger on an I/O signal or on an interrupt input that can occur on any clock cycle.

Choose the Trigger window Options menu Bus or Clock item.

trigInputMode=[activeHigh | activeLow] specifies whether the EA Trigger window Ext condition matches a high or low Trigger In signal. Choose the Trigger window Options menu Trigger In High or Low item.

trigOutputMode=[activeHigh | activeLow | openCollector] specifies the EA Trigger window Ext action Trigger Out signal value. Choose the Trigger window Options menu Trigger Out Active High, Low, or Open Collector item.

powerpak.ini lines specifying Trigger options

```
[TrigInfo]
// The following are the installed default values.
numTraceBuffers=1
traceAlignment=pre
breakOnFull=off
counterTimer=counter
trigMode=bus
triggerInActive=low
triggerOut=activeLow
```

[VariableInfo]

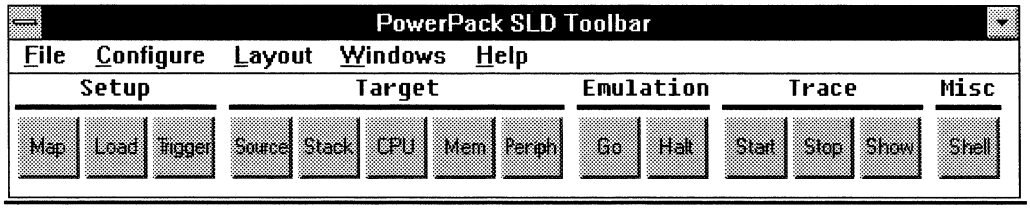
supports bitfield types

AutoCalcBitfieldOffsets=[1 | 0] specifies whether to calculate the SLD software bitfield offsets automatically. Set this entry to **1** when the toolchain does not generate bitfield member offsets.

powerpak.ini lines resolving toolchain specifics

```
[VariableInfo]
// The following is the installed default value.
AutoCalcBitfieldOffsets=0
```

Toolbar Reference



The Toolbar opens when you start the SLD software and is always available. Options unavailable for your emulator configuration are greyed-out. Closing the Toolbar ends your emulator session. Minimizing the Toolbar hides all other SLD windows and icons.

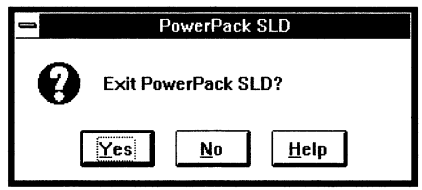
Toolbar Menus

Menu	Use To:
File	Exit the SLD software.
Configure	Configure and initialize the debugging environment.
Layout	Save your screen layout of SLD windows.
Windows	Select a closed or iconized SLD window to open.
Help	Open a window for help with the SLD software.

File Menu

You can exit the SLD software as you would exit any Windows application; or you can open the File menu and choose Exit. The emulator asks you to confirm exiting.

Exit dialog box, popped-up from the Toolbar File menu, to exit from the SLD software

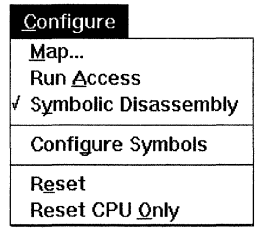


In any SLD window other than the Toolbar, choosing Exit closes only that window. Exit is on every SLD window File menu except in the CPU window, where Exit is on the Options menu.

Configure Menu

Configure menu items vary between processors.

Toolbar Configure menu



Map... opens the Map dialog box for examining and modifying your memory map. Choosing this menu item has the same effect as choosing the Map button. The Map dialog box is described in the "Map Dialog Boxes" section later in this chapter. You can also configure memory with Map and RestoreMap Shell commands.

Run Access, when checked, enables memory access during emulation. Memory access is used to update the Peripheral and Memory windows and to read or write peripheral registers and memory. (Run access does not affect CPU register access, which is always unavailable during emulation.) Because memory access takes a small amount of processor time, doing such operations during emulation can degrade your program performance. Initially, run access is disabled (unchecked) and memory access is available only when emulation is halted.

You can also enable and disable run access with the RunAccess Shell command.

Symbolic Disassembly, when checked, uses symbolic addresses in the disassembly displayed in the Source and Memory windows.

Save Chip Selects... records the chip-select register values in an ASCII file. For a list of saved registers, see the *Hardware Reference*. The values can be restored with the Restore Chip Selects item.

You can also save the chip select registers with the SaveCS Shell command.

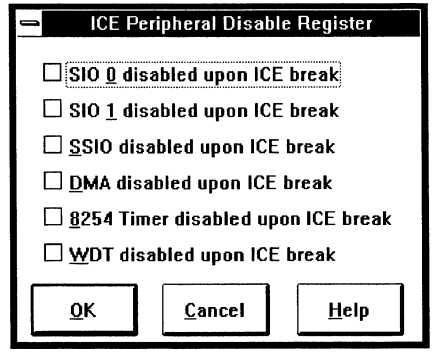
Restore Chip Selects... restores the chip-select register values from an ASCII file. You can create this file with the Save Chip Selects item, with a SaveCS Shell command, or with a text editor.

Configure Symbols updates the loaded symbols with the base address from the descriptor table (GDT or LDT). Your program must provide the GDTR and LDTR values and the GDT and LDT contents.

ICECFG0 Register... opens the ICE Peripheral Disable Register dialog box for setting bits in the Intel386 EX processor ICECFG0 register. To enable or disable specific peripherals on ICE break, check or uncheck

each option. The following shows an ICE Peripheral Disable Register dialog box with all peripherals enabled on ICE break.

ICE Peripheral Disable Register dialog box, accessed from the Toolbar Configure menu ICECFG0 item, with all peripherals enabled on ICE break



Reset resets and reinitializes the target processor:

- The processor RESET pin is asserted.
- The program counter is read from memory; the Source window is scrolled to the beginning of code.
- The stack pointer is read from memory, resetting the stack; the Stack window display becomes invalid.
- All SLD windows are updated.

You can also reset the processor with the Source window Run menu Reset item, the CPU window Options menu Reset item, or the Reset Shell command.

Reset CPU Only resets only the processor and does not update the windows. Use Reset CPU Only if Reset fails to reset the processor.

You can also reset only the the processor with the CPU window Options menu Reset CPU Only item or the Reset Shell command.

Layout Menu

Save Settings Now saves the SLD screen layout immediately.

Save Settings On Exit saves SLD screen layout when you exit.

Toolbar Buttons

Button	Use To:
Map	Open the Map dialog box (described later in this chapter) to examine or change the memory configuration. This button has the same effect as the Configure menu Map item. You can also configure memory with the Map and RestoreMap

Shell commands.

Load — Open the Load dialog box (described later in this chapter) to load code and/or symbols. You can also load code and symbols with the **Load** Shell command or the Source window File menu Load Code item.

Trigger — Open the Trigger window to define triggers and events for controlling emulation and trace collection. This button has the same effect as the Windows menu Trigger item. (PP and EA only)

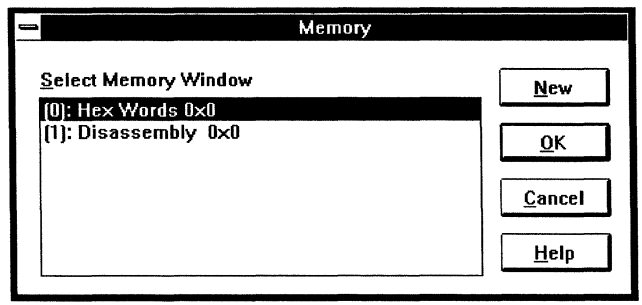
Source — Open the Source window to examine source and disassembly, manage breakpoints and stepping, and find source corresponding to displayed trace. This button has the same effect as the Windows menu Source item.

Stack — Open the Stack window to view the current nested calls, associated parameters and variables, and stack usage statistics. This button has the same effect as the Windows menu Stack item. You can also examine the stack with the **StackInfo** and **StackArea** Shell commands, or modify the stack with the **StackArea**, **StackBase**, and **StackSize** Shell commands.

CPU — Open the CPU window to view and change processor registers. This button has the same effect as the Windows menu CPU item. You can also display and edit the CPU registers with the **Register** Shell command.

Mem — Open or change focus to one of up to 20 Memory windows to view and change memory. This button has the same effect as the Windows menu Memory item. You can also view and change memory with the **Dump**, **Write**, **Fill**, **Search**, and **Copy** Shell commands. If more than one Memory window (including minimized windows) is open, a dialog box appears for choosing an existing Memory window or open a new one.

Memory window selection dialog box, accessed from the Toolbar Mem button when multiple memory windows are open

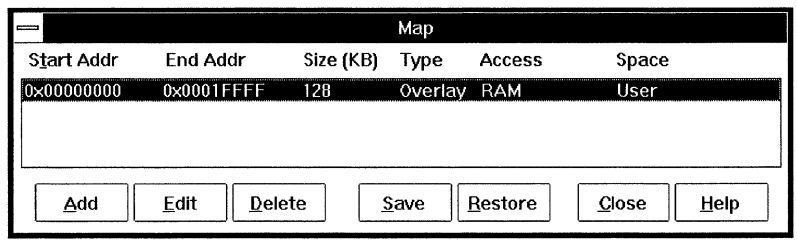


Periph	Open the Peripheral window to view and change peripheral register values. This button has the same effect as the Windows menu Peripheral item. Peripheral registers are unavailable on some processors.
Go	Start emulation from the current program counter, controlled by previously defined breakpoints and triggers. This button has the same effect as the <F9> key, the Source window Go button and Run menu Go item, and the Shell Go command.
Halt	Stop emulation. This button has the same effect as the <F2> key, the Source window Halt button and Run menu Halt item, and the Shell Halt command.
Start	Begin collecting trace. Tracing starts automatically when emulation starts. You can start and stop trace collection during emulation without affecting emulation. You can also start trace with the Trace window Trace menu Start item.
Stop	Stop collecting trace. You can also stop trace with the Trace window Trace menu Stop item.
Show	Open the Trace window to display collected trace. You can examine trace during emulation. This button has the same effect as the Windows menu Trace item.
Shell	Open the Shell window for command-line entry. This button has the same effect as the Windows menu Shell item.

Map Dialog Boxes

The Map dialog box lists the configuration of each mapped region. To select a region, click on it or use the <Up Arrow> and <Down Arrow> keys to move the highlight.

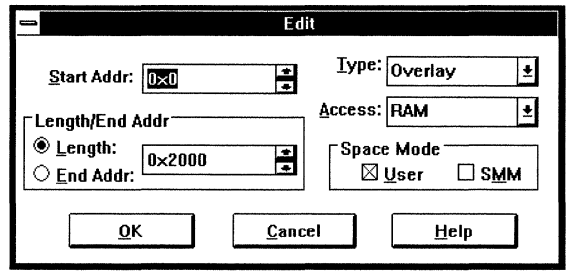
Map dialog box with 128K bytes of overlay memory mapped for RAM access



Map Dialog Box Buttons

Button	Use To:
Add	Open a dialog box to configure unmapped memory. Valid Start Addr and Length/End Addr values depend on how much memory is available.

Map Edit dialog box, similar to the Map Add dialog box, accessed from the Map dialog box Edit button



For more information on the Start Addr, Length/End Addr, and Access field values, see the list of Map dialog box field contents below.

Edit	Open a dialog box (see the Add button description above) to reconfigure a mapped region. This button is available when a listed region is selected.
Delete	Revert a mapped region to unmapped memory. This button is available when a listed region is selected.
Save	Open a dialog box to save the listed configuration to a map (*.map) file. You can also use the SaveMap Shell command to save the map configuration.
Restore	Open a dialog box (see the Save button description above) to configure regions from a previously saved map (*.map) file. You can also use the RestoreMap Shell command to restore a previously saved map configuration.
Close	Close the Map dialog box.
Help	Open a window for help on mapping.

You can also use the Map Shell command to examine and modify memory mapping.

Map Dialog Box Fields

Field	Value
Start Addr	must start on a 4K boundary.
End Addr	can end on any address.
Size	varies between processors:

- for PP-386 and SW-386 emulators, any multiple of 4K bytes starting on any 4K address
- for EA-486 emulators, any multiple of 128K bytes starting on any 128K address
- for EA-NS486 emulators, any multiple of 4K bytes starting on any 128K address

Specify a region size instead of an end address by choosing the Length rather than the End Addr button in the Map Add/Edit dialog box, then filling-in an appropriate value in the Length/End Addr field.

Type — is Overlay or Target. You can install 1M or 4M bytes of overlay memory on the emulator to substitute for target memory. To use the overlay memory, you must map a region as Overlay. Unmapped regions are mapped as Target and use your target board memory.

Access Rights — is one of the following ways to control and alert you to memory access by your program:

RAM allows read and write access.

ROM BREAK (Intel processors only) allows read access; prevents write access; and breaks on attempted write access. For Target memory, write access is allowed but causes emulation to break.

ROM NOBREAK allows read access; prevents write access; does not break on attempted write access. For Target memory, write access is allowed.

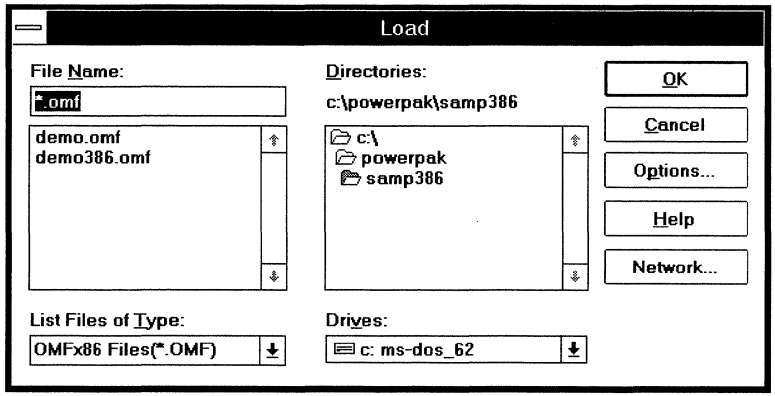
NONE (Intel processors only) prevents any access; breaks on attempted access. For Target memory, read and write accesses are allowed but cause emulation to break.

Space — (Intel processors only) is User or SMM (system management mode).

Load Dialog Boxes

Open a dialog box for loading code and symbols with the Toolbar Load button.

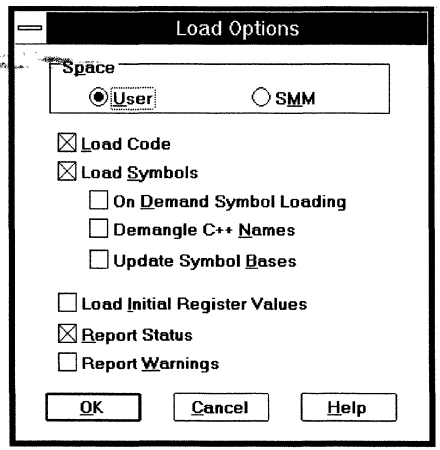
Load dialog box, accessed from the Toolbar Load button



When you select a loadfile, the Options button in the Load dialog box becomes available. Choosing this button opens the Load Options dialog box for specifying how to load code and/or symbols from the loadfile. Available options depend on your processor and loadfile format.

When you are ready to load, choose the OK button. To exit the Load dialog box without loading, choose the Cancel button. To open a window with help on loading, choose the Help button.

OMF386 Load Options dialog box, accessed from the Load dialog box Options button



Be sure the space you select is compatible with the address space configured in the Map dialog box. (Intel processors only)

To enable an option, check the box beside the option. To disable an option, uncheck the box.

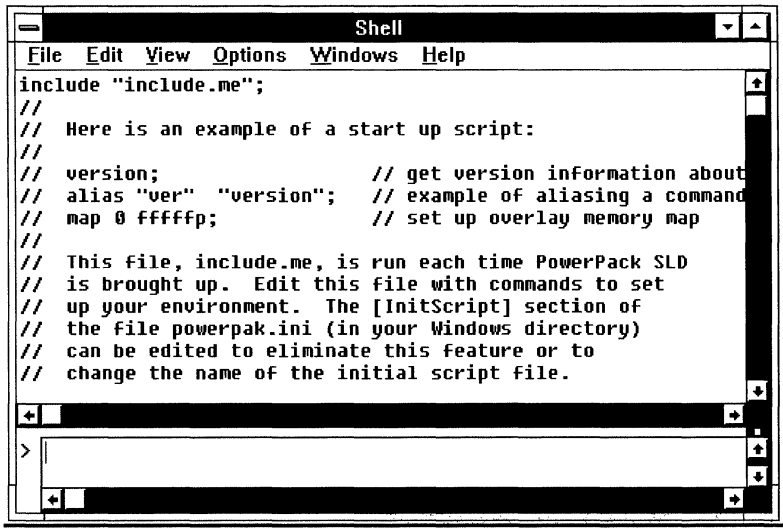
Option	Effect
Load Code	loads executable code sections from your loadfile.
Load Symbols	loads data sections and relevant symbolic

	information from your loadfile. When this option is enabled, several sub-options are available.
On Demand Symbol Loading	waits to load symbolic information for each module until it is needed, for example when you display the module in the Source window.
Demangle C++ Names	uses an MRI algorithm to demangle some C++ symbols, for example overloaded function names.
Update Symbol Bases	reads base addresses for symbol tables, once the registers are initialized with Load Initial Registers.
Load Initial Register Values	initializes the processor registers from information put into the loadfile during compilation and linking.
Report Status	displays an information box showing the load operation progress.
Report Warnings	displays information boxes with non-fatal anomolies encountered during loading.

You can load a file during emulation. Be sure the file's load addresses do not overlap the memory occupied by the running program. Loading a file at a location in use stops the emulator in an unpredictable state.

You can specify equivalent load options with the Load Shell command.

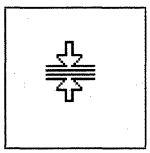
Shell Window Reference



Shell Window Contents

The Shell window contains two panes:

Transcript	in the top part of the window, echoes commands and command output.
Command Entry	in the bottom part of the window, is where you enter commands.



You can change the relative sizes of the Shell window panes. A split box between the vertical scroll bars defines the edge between the Transcript and Command Entry panes. When the mouse is pointing to the split box, a split-box cursor appears (see figure at left). Drag the split box to resize the panes.

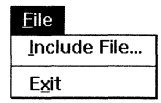
To change focus from one pane to the other, click in the inactive pane or press the <Tab> key.

Shell Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

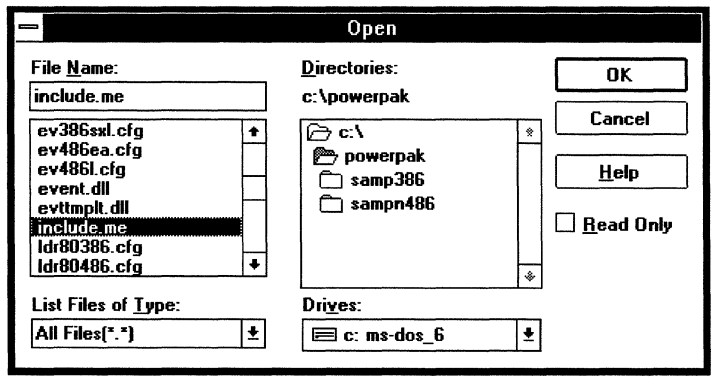
Menu	Use To:
File	Run a script; close the Shell window.
Edit	Manage text in the Command Entry and Transcript pane using Windows Clipboard.
View	Manage the Transcript pane display.
Options	Manage log files, command history, and the Transcript size.
Help	Open a window for help with the SLD software.

File Menu



Include File... opens a dialog box wherein you can select a script (a text file of Shell commands) to be run immediately.

Open dialog box to run a Shell script, with the include.me script selected



Exit closes the Shell window without exiting the SLD software.

Edit Menu

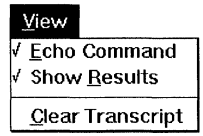
Cut moves highlighted strings to the Windows Clipboard.

Copy copies highlighted strings to the Windows Clipboard.

Paste copies strings from the Clipboard to the Command Entry pane.

View Menu

View menu with Echo Command and Show Results enabled



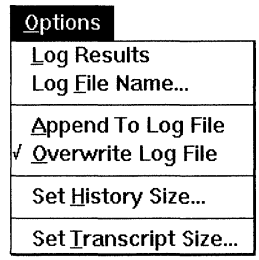
Echo Command displays in the Transcript pane all commands you enter in the Command Entry pane.

Show Results displays in the Transcript pane the results of commands you enter in the Command Entry pane.

Clear Transcript blanks the Transcript pane.

Options Menu

Options menu with Overwrite Log File enabled



Log Results starts recording into a text file all that appears in the Transcript pane. If you have not previously specified a log filename, the emulator uses shell.log in your SLD directory (c:\powerpak\shell.log if you installed to the default directory).

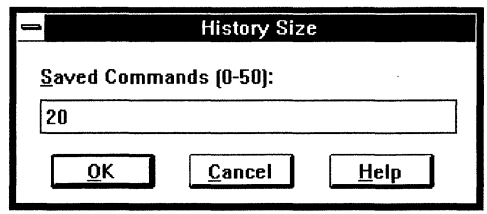
Log File Name... opens a dialog box to specify the logfile pathname.

Append To Log File ensures that text recorded into an existing file is added to the end of the file without destroying any prior file contents.

Overwrite Log File ensures that text recorded into an existing file overwrites the file, destroying any prior file contents.

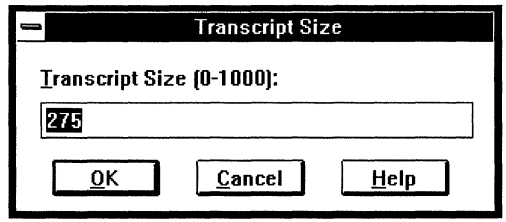
Set History Size... opens a dialog box to specify the maximum number of commands retained in the history buffer. Recall past commands with <Ctrl><Up Arrow> and <Ctrl><Down Arrow> key combinations.

History Size dialog box, specifying that a running history of the 20 most recent Shell commands be kept



Set Transcript Size... opens a dialog box to specify the maximum number of lines retained in the scrollable Transcript pane.

Transcript Size dialog box, specifying that the 275 most recent lines of Shell command activity be kept



Entering Commands in the Shell Window

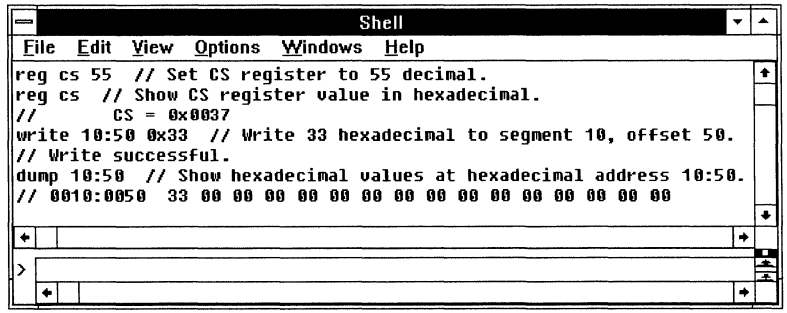
Enter commands in the Shell window by one of:

- Type a command. Press <Enter> to execute it.
- Type a sequence of commands. Follow each command with a semicolon (;). Press <Ctrl><Enter> to start a new line without executing the already typed commands. Press <Enter> to execute the sequence of commands.
- Execute a script (a text file of commands delimited by semicolons). You can create or change a script in a text editor. To execute a script, use the Shell window File menu Include item or the Include command (described later in this chapter). In the powerpak.ini file, you can specify a script to execute automatically when the SLD software starts. The default initialization script is include.me.
- Recall previously entered commands from the history buffer by scrolling with <Ctrl><Up Arrow> or <Ctrl><Down Arrow>. Edit the command line as needed, then press <Enter> to execute. To specify the history buffer size, fill-in the Options menu Set History Size dialog box.

To cancel a command line without executing it, press <Esc> instead of <Enter>. To interrupt command execution, press <Esc>.

The emulator interprets addresses as hexadecimal and data as decimal values. Prefix hexadecimal data with 0x, as shown in the following:

Shell commands with hexadecimal addresses, decimal data, and hexadecimal data



Shell Window Commands

Notational Conventions

The following notational conventions are used in the following pages:

Notation	Meaning
COMMANDNAME commandname CommandName	Case is not significant in command names, keywords, and aliases. Case is significant in Shell variables.
<placeholder>	Indicates an symbol or expression argument.
[item]	Brackets delimit an argument that can be entered no more than once.
(item)	Parentheses delimit an argument that must be entered at least once.
{item}	Braces delimit an argument that can be entered zero or more times.
item1 | item2	A vertical bar separates mutually exclusive arguments.
"<string>"	Delimit string constants with double quotes.
/* comment */ // comment	Delimit comments C-style or C++-style.
// response	Forward slashes precede command output.

Commands and System Variables Grouped by Function

To Do	Use	For
Address translation	Xlt	translating numeric and symbolic address formats
Assembly; disassembly	Asm	assembling lines of code directly into memory
	AsmAddr	determining the location and address mode for assembling into memory
	Dasm	showing memory contents as disassembly
	DasmSym	showing symbolics in disassembled memory
Breakpoints	Bkpt	setting and showing breakpoints
	BkptClear	removing breakpoints
	DR	managing debug registers
Bus error management	BusRetry	managing bus contention and timeout
Chip select setup	RestoreCS	setting 386EX and NS486 chip select values from a file
	SaveCS	saving 386EX and NS486 chip select values to a file
Compiler setup	MaxBitFieldSize	setting the bit field limit for OMF386 loadfiles
CPU data structures	DT	displaying the descriptor table
	GDT	displaying the global descriptor table
	GetBase	displaying the symbol base
	IDT	displaying the interrupt descriptor table
	LDT	displaying the local descriptor table

Commands and System Variables Grouped by Function (continued)

To Do	Use	For
	PD	displaying the page directory
	PMode	displaying the address mode
	TSS	displaying the task state segment
Emulation	Go	emulating
	GoInto	emulating until a function call or return has occurred
	GoUntil	emulating until just before a function call or return
	Halt	halting emulation
	ResetAndGo	resetting the processor, then emulating
	Step	emulating one or more instructions
	StepSrc	emulating one or more source statements
Event definition	EventRestore	setting event definitions from a file
	EventSave	saving event definitions to a file
Help	Help	invoking SLD on-line help
Loading	Load	loading code and symbols
	LoadSize	determining the memory access size for loading
	ResetLoaders	correcting an internal loader error on request
Memory management	Copy	copying contents between memory locations
	Dump	disassembling memory to the Transcript pane

Commands and System Variables Grouped by Function (continued)

To Do	Use	For
	Fill	writing a repeating pattern to memory
	Map	setting and showing memory access options
	RestoreMap	setting map information from a file
	RunAccess	allowing memory access during emulation
	SaveMap	saving the map to a file
	Search	finding a value or pattern in memory
	Size	determining the memory access size
	Verify	checking memory writes
	Write	writing a value to memory
Register access	Config	configuring the 386EX HLDA signal
	Register	reading or writing CPU register values
Resetting processor	Reset	resetting the processor and SLD windows or the target
	ResetAndGo	resetting the processor, then emulating
Shell commands	Alias	defining one string to be substituted for another
	Append	adding new log information to existing log
	Clear	erasing the Transcript pane
	Delete	removing a Shell variable, alias, or link
	Echo	toggling command display in the Transcript pane

Commands and System Variables Grouped by Function (continued)

To Do	Use	For
	Exit	exiting the Shell window
	History	sizing the history buffer
	If	executing Shell commands conditionally
	Include	executing a script
	Integer	finding whether a Shell variable is an integer
	Link	managing source filenames
	List	showing Shell variables
	Log	opening a log file
	Logging	starting or stopping the log
	Overwrite	replacing previous log with new log information
	Print	showing Shell variables with specified text
	Results	toggling response display in the Transcript pane
	String	discovering whether a Shell variable is a string
	Transcript	setting or showing the Transcript pane size
	Time	showing the current date and time
	While	repeatedly executing Shell commands conditionally
Stack management	DisableAlarmLimit	disabling the stack usage alarm
	DisableHighWaterMark	disabling the stack maximum-usage indicator
	DisplayStack	showing the stack contents

Commands and System Variables Grouped by Function (continued)

To Do	Use	For
	EnableAlarmLimit	enabling the stack usage alarm
	EnableHighWaterMark	enabling the stack maximum-usage indicator
	FillStackPattern	writing a repeating value to the stack area
	SetStackAlarm	specifying the stack usage alarm
	SetStackArea	determining the stack base and size
	SetStackBase	determining the stack base
	SetStackSize	determining the stack size
	StackInfo	showing the stack definition and statistics
Status reporting	$BREAKCAUSE	showing why emulation halted
	$EMULATING	showing whether emulation is halted
	$PROCESSOR	identirying target CPU
	$PROCFAMILY	identifying target CPU family
	$PROCTYPE	identifying target CPU type
	$SHELL_STATUS	showing the last Shell command status
	$SYSTEMTYPE	identifying emulator and probe CPU
	BusRetry	managing bus contention and timeout
	Cause	showing why emulation halted
	EmuStatus	showing current emulator activity

Commands and System Variables Grouped by Function (continued)

To Do	Use	For
	IsEmuHalted	showing whether emulation is halted
	Time	displaying the current time
	Version	displaying host and emulator version information
Symbol management	AddressOf	displaying a symbol's load address
	ConfigSymbols	updating the symbol base from registers
	DisplaySymbols	displaying program symbolic information
	NameOf	displaying the symbol associated with an address
	RemoveSymbols	removing loaded symbolic information
	SetBase	setting the symbol base
	SymbolCloseFile	closing the file of recorded symbolic information
	SymbolOpenFile	recording symbolic information to a file
Target control	Reset	resetting the processor or the target system
	Signal	determining whether signals are driven by the emulator or the target
Testing hardware	RAMtst	running the memory confidence tests
	Test	running the hardware confidence tests
Timing	LapTimer	displaying the timer
	StartTimer	starting the timer

Commands and System Variables Grouped by Function (continued)

To Do	Use	For
	StopTimer	stopping the timer
Tracing	Flush	flushing the cache

Command Dictionary

$BREAKCAUSE

System variable; shows what caused emulation to break.

Related topics: $EMULATING, Cause, Go, GoInto, GoUntil, Halt, ResetAndGo, Step, StepSrc

$BREAKCAUSE

Case is significant. Enter this variable in upper case.

Knowing what caused emulation to break can be useful; for example, a script can single-step repeatedly until the target processor is reset.

$BREAKCAUSE is updated when emulation breaks. Its value indicates the cause of the break:

0 No cause (for example, emulation not yet started)

1 Target processor was reset

2 Emulator was halted

4 Processor single step

5 Execution breakpoint reached

8 External break request

9 Unknown cause

```
/* Following is part of a script that stops after any execution
breakpoint. $Z is an undeclared Shell variable that stops the
script. */

go;
while ($EMULATING) {;};     /* loop until emulator halts */
if ($BREAKCAUSE==5) {$Z;}; /* test for execution breakpoint */
```

$EMULATING

System variable; shows whether the emulator is running.

Related topics: $BREAKCAUSE, Cause, Go, GoInto, GoUntil, Halt, ResetAndGo, Step, StepSrc

$EMULATING

Case is significant. Enter this variable in upper case.

Knowing whether the emulator is running can be useful, for example, to control script execution flow based on emulation status.

$EMULATING has the value:

1 The emulator is running.

0 The emulator is halted.

```
bkpt #main;                    /* stop after registers initialized */
ResetAndGo;                    /* start from the power-on level */
while ($EMULATING) {;};          /* loop until emulator halts */
```

$PROCESSOR

System variable; identifies target processor.

Related topics: $PROCFAMILY, $PROCTYPE, $SYSTEMTYPE, Version

$PROCESSOR

Case is significant. Enter this variable in upper case.

$PROCESSOR identifies the processor in your target design as:

Value	Processor
386cx	Intel386 CX
386dx	Intel386 DX
386ex	Intel386 EX
386sx	Intel386 SX
3exc	Intel386 EX C-step
486	Intel386 CX
486dx	Intel386 EX
486sx	Intel386 SX
ns486	National Semiconductor NS486SXF
none	No processor specified

$PROCFAMILY

System variable; identifies target processor family.

Related topics: $PROCESSOR, $PROCTYPE, $SYSTEMTYPE, Version

$PROCFAMILY

Case is significant. Enter this variable in upper case.

$PROCFAMILY has the value:

Value	Processors in Family
FAMILY_X86	Intel386, Intel486, or NS486SXF

$PROCTYPE

System variable; identifies target processor type.

Related topics: $PROCESSOR, $PROCFAMILY, $SYSTEMTYPE, Version

$PROCTYPE

Case is significant. Enter this variable in upper case.

$PROCTYPE identifies the processor type in your target design as:

Value	Processors Categorized as This Type
80386	Intel386 EX, CX, or SX
80486	Intel486 SX or NS486SXF

$SHELL_STATUS

System variable; shows whether the last shell command completed successfully.

$SHELL_STATUS

Case is significant. Enter this variable in upper case.

Knowing whether a Shell command completed successfully can be useful, for example, to control script execution flow based on whether prior commands executed correctly.

$SHELL_STATUS has the value:

0	The command completed normally.
nonzero	An error occurred. The $SHELL_STATUS value is the SLD software error code.

```
bkpt #main;                          /* stop after registers initialized */
Reset;          /* try to reset processor and update SLD windows */
If ($SHELL_STATUS) {
  Print "Didn't Reset";
  Reset CPUonly};        /* Reset without updating SLD windows */
```

$SYSTEMTYPE

System variable; identifies emulator and probe processor.

Related topics: $PROCESSOR, $PROCFAMILY, $PROCTYPE, Version

$SYSTEMTYPE

Case is significant. Enter this variable in upper case.

$SYSTEMTYPE identifies your emulator as:

Value	Emulator
PP386cx	PP emulator for the Intel386 CX processor
PP386dx	PP emulator for the Intel386 DX processor
PP386sx	PP emulator for the Intel386 SX processor
LC386ex	SW emulator for the Intel386 EX processor
LC3exc	SW emulator for the Intel386 EX C-step processor
LC486	EA emulator for the Intel486 processor
LC486dx	EA emulator for the Intel486 DX processor
LC486sx	EA emulator for the Intel486 SX processor
LCns486	EA emulator for the NS486SXF processor

AddressOf

Returns the numeric address of a module, function, line, or variable.

Related topics:
DisplaySymbols, GetBase, NameOf, RemoveSymbols, SetBase

AddressOf <address>

<address> is a partly or fully qualified symbol name.

AddressOf returns the numeric address where the symbol is loaded.

For local variable addresses (stack offsets), use DisplaySymbols. You cannot use AddressOf to obtain the address of a local variable, because a local variable has no fixed location.

```
addressof #Blank_TxBuf;                    // address range of a function
// 6A6..6BF
addressof #MsgRx;                          // address range of an array variable
// E68..E87 [32]
```

For function names, you can obtain the same information in the Source window by double-clicking on the function name to display the Function pop-up menu, then choosing Show Load Address.

Alias

Define or list an alias.

Related Topics:
Delete

Alias ["<name>" ["<value>"]]

<name> is the alias. The quotation marks are required.

<value> assigns a value to the specified name. The quotation marks are required. Inside <value>, replace double quotation marks with single quotation marks.

With no arguments, Alias lists all currently defined aliases. Alias "<name>" displays the value of <name>.

Use alias to shorten or change commonly used command strings.

```
alias "s1" "include 's1.inc'";
Alias "increment" "$a = $a + 1; $a;"
$a = 0;
increment;
// 0x1 1
increment;
// 0x2 2
```

Append

Appends to log file.

Related topics:
Log, Logging, Overwrite, Echo, Results

Append

When Append has been specified, logging adds text to the end of the current log, preserving the log's prior contents.

You can also configure logging to append to a file with the Shell window Options menu Append To Log File item.

```
Echo On;                    // Commands you enter appear
                            // in the Transcript pane.
Results On;                 // Results of the commands appear
                            // in the Transcript pane.
Append;                     // Subsequent logging will add
                            // to any prior log contents.
Log "emu1.log";             // Open the log file emu1.log.
Logging On;          // Start writing log information. The emulator
                     // immediately puts the time and date in the log file.
//...                       // Your emulation activities...
Logging Off;         // Stop writing log information. The emulator
                     // immediately puts the time and date in the log file.
```

Asm

Write assembly to memory.

Related topics:
AsmAddr, Dasm, DasmSym

Asm <string>

<string> is an assembly language statement.

Asm checks the syntax of <string> and writes the instruction bytes to memory at the current assembly address. (Determine the current assembly address with AsmAddr.)

Symbolic assembly is not supported.

```
Asm nop;
// 000000 4E71          nop
// Number of bytes: 2
```

You can also assemble new instructions and data into memory with the Single-Line Assembler dialog box.

AsmAddr

Set the address where the Asm command will write.

Related topics:
Asm, Dasm, DasmSym, Pmode

AsmAddr [<mode>] [<address>] [<space>]

<mode> specifies the addressing mode:

- Auto derives the addressing mode based on Pmode.
- Use16 uses 16-bit operands and addresses.
- Use32 uses 32-bit operands and addresses.

<address> is a numeric or symbolic address of the location where the next Asm command will write.

<space> specifies the emulator address space as:

- user, smm, or io for 386 EX emulators
- user or smm for 386 CX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

With no arguments, AsmAddr displays the current assembly address in the current addressing mode.

```
AsmAddr 2000;
// Asm address offset: 2000
```

Bkpt

Display, set, or modify breakpoints.

Related topics:
BkptClear, DR

Bkpt [enable | disable] [temporary | permanent] [<address>] [@<ID>] [<space>]

enable with @<ID> specified, enables the breakpoint; otherwise enables all breakpoints.

disable with @<ID> specified, disables the breakpoint; otherwise disables all breakpoints.

temporary removes the breakpoint when the breakpoint halts emulation.

permanent retains the breakpoint when the breakpoint halts emulation. To remove the breakpoint, explicitly delete it.

<address> a numeric or symbolic address. When this address is accessed, the breakpoint (if enabled) halts execution.

<ID> is an integer from 0 to 65534 uniquely identifying the

breakpoint. Either you or the emulator assigns an ID when a breakpoint is defined. Specifying an existing ID modifies the identified breakpoint. The at (@) is required.

<space> for 386 EX, 386 CX, or Intel486 SLE emulators specifies smm or user (the default) address space.

With no arguments, Bkpt displays all current breakpoints. Source information is also displayed when a match exists with the symbol table.

```
bkpt disable @12          /* disable the breakpoint with ID 12 */
```

You can also set breakpoints using the Source window mouse or Breakpoints menu, or the Breakpoint window Set button or Breakpoints menu.

BkptClear

Remove breakpoints.

Related topics: Bkpt, DR

BkptClear @<ID> | <address> [<space>] | all

<ID> removes the breakpoint with the specified ID number. The at (@) is required.

<address> removes the breakpoint at the specified code address.

<space> for 386 EX, 386 CX, or Intel486 SLE emulators specifies smm or user (the default) address space.

all removes all temporary and permanent breakpoints.

Use BkptClear to remove a specified breakpoint or all temporary and permanent breakpoints.

```
BkptClear @1;             /* remove breakpoint with id 1 */
BkptClear all;                  /* remove all breakpoints */
```

You can also clear breakpoints with the Source window mouse or Breakpoints menu, or the Breakpoint window Clear button or Breakpoints menu.

BusRetry

Assert bus error after timeout.

BusRetry [on | off]

on turn retry on.

off turn retry off.

With no arguments, BusRetry displays its current setting.

Disable retry when contention exists with another driver or when a slow device takes longer than the timeout.

Cause

Display the cause of the last break in emulation.

Related topics:
$BREAKCAUSE

Cause

Use this command when emulation is halted to discover the reason for the most recent halt. Possible Cause responses are:

- No cause is recorded.
- The target processor was reset.
- You entered a Halt command.
- The emulator completed a Step.
- Emulation encountered an execution breakpoint.
- The emulator received an external break request.
- The cause is unknown.

The break cause also appears in the Status window.

Clear

Clear the Shell window Transcript pane.

Clear

Use Clear to remove all text from the Shell window Transcript pane. The Shell window View menu Clear Transcript item does the same.

Config

Define Intel386 EX HLDA pin function.

Config ignoreHLDA [on | off]

on causes the emulator to ignore the HLDA pin state. Set config ignoreHlda on when HLDA is programmed as an I/O bit.

off (default) causes the emulator to examine the HLDA pin state before generating overlay RAM or trace/trigger strobe.

With no arguments, Config displays its current setting.

On the 386 EX, you can program the HLDA pin to function either as HLDA function or as an I/O bit. The emulator hardware must know when the bus has been granted to an external master so that overlay RAM cycles are disabled to prevent corruption. If the HLDA pin is visible, the emulator disables overlay RAM cycles. Otherwise, the emulator assumes

no external masters exist.

When using the Intel Evaluation Board, which programs the HLDA pin to be an I/O bit, set config ignoreHlda on.

ConfigSymbols

Update symbol base address from the x86 descriptor table.

ConfigSymbols [<base>]

<base> is the base name for the group of symbols to be updated.

With no arguments, ConfigSymbols reconfigures all symbols in your program.

This command updates the specified symbols with the base address obtained from the descriptor table (either GDT or LDT). To get the correct symbol base, the target program must set up the correct values of GDTR and LDTR and the contents of those tables.

You can also update the symbol base address with the Toolbar Configure menu Configure Symbols item.

Copy

Copy one region of target or overlay memory to another.

Related topics: Dump, Fill, RunAccess, Search, Size, Verify, Write

Copy <start> (<end> | Length <len>) [<space>] [Target]
To (<dest> | Target | <dest> Target) [<space>]

<start> specifies the starting address of the region to be copied.

<end> specifies the ending address of the region to be copied.

<len> specifies the number of bytes to be copied. The Length keyword is required.

<space> specifies user (the default) or:

- smm for 386 EX, 386 CX, or Intel486 SLE emulators
- io for NS486 emulators

Target overrides any overlay mapping to use target memory as the source or destination.

<dest> Specifies the starting address that will be copied into. The To keyword is required.

Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Use RunAccess to enable Copy during emulation; however, such access can degrade your program execution.

```
/* Copy 64 KB from address 0x0 to overlay at the same address: */
```

```
map 0 10000;
copy 0 length 1000 target to 0;
/* Copy from overlay to target: */
copy 0 length 1000 to 0 target;
/* Copy from overlay to overlay: */
copy 1000 length 1000 to 4000;
/* Use symbolic addresses: */
copy #func1 #func2 to #ram_area target;
```

You can also copy memory with the Memory window Edit menu Copy Memory item.

Dasm

Disassemble memory.

Related Topics:
AsmAddr
DasmSym

Dasm [<mode>] [<start> [<end>] [<space>]]

<mode> Specifies the addressing mode:

- Auto derives the addressing mode based on the pmode.
- Use16 uses 16-bit operands and addresses.
- Use32 uses 32-bit operands and addresses.

<start> is the first address of the region to disassemble.

<end> is the last address of the region to disassemble.

<space> for 386 EX, 386 CX, or Intel486 SLE emulators specifies smm or user (the default) address space.

With no arguments, 10 instructions are disassembled beginning at the current assembly address. (To find the current assembly address, use AsmAddr.) When only <start> is specified, 10 instructions starting at <start> are disassembled.

You can also view disassembled memory with the Memory window View menu Disassembly item, or interleaved in your source text with the Source window View menu Mixed Source And Asm item.

DasmSym

Control symbolic disassembly in the Shell window.

Related topics:
AsmAddr, Dasm

DasmSym [on | off]

on (default) turns on symbolic disassembly.

off turns off symbolic disassembly.

With no arguments, DasmSym displays the current setting.

Symbolic disassembly displays symbols in the disassembly shown in the

Memory window in Disassembly view, the Source window Mixed Source And Asm view, and the Trace window Instruction view.

You can also toggle symbolic disassembly with the Toolbar Configure menu Symbolic Disassembly item.

Delete

Delete a Shell variable or alias

Related Topics:
Alias

Delete (Alias "<name>" | <variable> | Link "<filename>")

<name> is the alias to be deleted. The Alias keyword and the quotation marks are required.

<filename> identifies a file link to be deleted. The Link keyword and the quotation marks are required.

<variable> is the Shell variable to be deleted.

```
$a = $b = 0;

list;
// $a = 0
// $b = 0

Delete $a

list
// $b = 0

Alias "a" "$a;" ;

Alias;
// a: "$a;"

Delete Alias "a";

Alias;
```

DisableAlarmLimit

Disable the warning message for excessive stack usage.

Related topics:
DisableHighWater-Mark,
DisplayStack,
EnableAlarmLimit,
EnableHighWater-Mark,
FillStackPattern,

DisableAlarmLimit

You can set an alarm (using EnableAlarmLimit) to notify you when stack usage exceeds a specified percentage of the stack. DisableAlarmLimit turns off this alarm.

You can also disable the alarm by un-checking the Stack window Options menu Enable Alarm Limit item.

SetStackAlarm, ,
SetStackArea,
SetStackBase,
SetStackSize,
StackInfo

DisableHighWaterMark

Disable keeping track of the stack maximum usage.

Related topics:
DisableAlarmLimit,
DisplayStack,
EnableAlarmLimit,
EnableHighWater-Mark,
FillStackPattern,
SetStackAlarm,
SetStackArea,
SetStackBase,
SetStackSize,
StackInfo

DisableHighWaterMark

You can set an indicator in the Stack window to keep track of the stack high-water mark (the maximum stack usage). DisableHighWaterMark turns off this indicator.

You can also disable the high-water mark by un-checking the Stack window Options menu Enable High-Water Mark item.

DisplayStack

Display the stack frames.

Related topics:
DisableAlarmLimit,
DisableHighWater-Mark,
EnableAlarmLimit,
EnableHighWater-Mark,
FillStackPattern,
SetStackAlarm,
SetStackBase,
SetStackSize,
StackInfo,
SetStackArea

DisplayStack [locals | hex]

locals	includes symbols for automatic variables.
hex	displays the stack in hexadecimal radix, 16 bytes per line.

When you specify no arguments, the display defaults to:

- addresses when no symbolic information is available
- addresses and function names when symbolic information is available

You can also view the stack frames, with stack and return addresses, arguments, and local variables, in the Stack window.

DisplaySymbols

Display all symbols or display one of the following: modules, functions, public symbols, or lines.

Related topics: AddressOf, GetBase, NameOf, RemoveSymbols, SetBase

DisplaySymbols [modules | functions | publics | lines | sorted | #<module>]

modules	lists module names only.
functions	lists modules, global variables, functions, and blocks.
publics	lists all printable symbols including publics (code labels and variables defined publicly across modules). For example, libraries normally contain no local symbols but accessible global variables in libraries appear as public symbols.
lines	follows each module by the line numbers loaded for that module. With each line number is listed the line's ending column and start address.
sorted	sorts the module list alphanumerically.
<module>	lists all symbols for the specified module. The hash mark (#) is required.

With no arguments, DisplaySymbols displays modules, global variables, functions, and local variables, but not publics nor individual line numbers.

If you have previously issued a SymbolOpenFile command, the DisplaySymbols output is directed to the symbol file.

The output is displayed in four columns:

- The symbol scope (MODULE, VARIABLE, FUNCTION, BLOCK, PUBLIC VAR, PUBLIC LABEL) appears in the first column. Each line is indented to show the level or scope of the symbol in the symbol hierarchy. Modules and publics are at the root level. Functions defined in a module are indented one level. Variables local to a function are indented under that function. Blocks are treated as unnamed functions and indented for each nesting level.
- The symbol name appears in the second column.
- The symbol type appears in the third column: the variable type; the function return type; the module source line number range; or the register description for a local register variable or argument.
- The symbol address appears in the fourth column. For static (fixed address) symbols, the address range in bytes appears followed by the decimal size of the range in square brackets ([<size>]). Local stack variable addresses are signed offsets from the stack frame pointer.

DR

Control debug register use.

DR [<num> Bkpt | User | [Data <mode> <address> <size> [Exact]]]

<num> identifies the debug register as 0, 1, 2, or 3.

Bkpt makes the register available for execution breakpoints.

User reserves the register for use by your program. The emulator avoids using this register for execution breakpoints and modifies DR7, allowing user access to any debug register.

Data configures the register as a data read/write breakpoint.

<mode> is one of:

- x sets the register to instruction execution mode. Emulation breaks on execution of the instruction starting at <address>.
- w sets the register to data write mode. Emulation breaks on a write to <address> in user space.
- rw sets the register to data read/write mode. Emulation breaks on a read or write to <address> in user space.

<address> specifies the virtual or linear base address of the breakpoint.

<size> specifies 1, 2, or 4 bytes starting with <address> as the address range of the data breakpoint. Emulation breaks on any data access completely or partly overlapping this range.

Exact ensures the processor waits after each instruction for all data cycles to complete. (Such waiting can degrade your program's performance.) A data breakpoint occurs immediately after the instruction that caused the breakpoint data cycle. (Execution breakpoints always occur exactly.) With exact not specified, several instructions can execute beyond the one that caused the breakpoint data cycle.

With no arguments, DR lists the debug register configurations.

When you set a breakpoint in the Source or Breakpoint window or with the Bkpt command, the emulator implements the breakpoint as either a DR or a software interrupt and as an execution or a data breakpoint. SLD installation configures DR[0..3] for execution breakpoints selected by the emulator and disables program access to DR7. To change this configuration, use DR to:

- Assign a specific execution or data breakpoint to each DR. A total

of four DR breakpoints can be concurrently defined, whether specified by you or by the emulator.

- Reserve each DR for program use, preventing the emulator from implementing a breakpoint in that register. Such reservation also enables undetected program access to system registers and DR7. Program changes to DR7 can cause unpredictable emulator behavior.

```
dr 0 user;                          /* Reserve dr0 for the target system. */
dr 1 bkpt;    /* Allow dr1 to be used as an execution breakpoint. */
dr 2;                        /* Show the current configuration of dr2. */
dr 3 data w 400L dword;        /* Define a double-word data write */
                               /* breakpoint at linear address 400. */
```

DT

Display descriptor tables.

Related topics: GDT, IDT, LDT, PD, TSS

DT (<selector> | <range> | <register> | Base <address> (<range> | Limit <bytes>)) [All]

<selector> specifies a selector.

<range> specifies the first and last of a range of selectors.

<register> is any CPU mnemonic specifying a register containing a selector in the first 16 bits.

<address> specifies the descriptor table base address. The Base keyword is required.

<bytes> specifies a range of selectors as a number of bytes. The Limit keyword is required.

All displays all entries, including invalid or reserved.

The descriptor table displayed for each selector is specified by the selector's bit 2 (TI).

```
dt 0x08 0x48 all;                   /* displays all entries */
                                    /* from selector 0x08 to 0x48 */
dt ds;                      /* displays the current ds descriptor entry */
```

Dump

Dump memory contents to the screen, formatted.

Related topics: Copy, Fill, RunAccess, Search, Size, Verify, Write

Dump [Loop] <addr1> [<addr2>] [Byte | Word | Long | Dword] [<space>]

<addr1> specifies the first address to be displayed. The address can be symbolic or numeric.

<addr2> specifies the last address to be displayed. Omitting <addr2> displays 16 bytes. The address can be symbolic or numeric.

Byte displays byte values.

Word displays word values.

Long displays double word values.

Dword is the same as Long.

<space> specifies the address space as:

- user, smm, or io for 386 EX emulators
- user or smm for 386 CX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

Loop repeatedly preforms the operation but prints no output to the screen, even if errors occur.

The physical read uses the Size command settings rather than the format size set by Dump. For example, if Size=Byte when a Dump command specifies Word, the emulator reads a set of byte-sized values and reformats them to display as word-sized values.

Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Use RunAccess to enable Dump during emulation; however, such access can degrade your program execution.

You can also view memory contents in up to 20 simultaneously active Memory windows as hexadecimal or decimal bytes, words, or dwords with equivalent ASCII characters; or as disassembled instructions.

Echo

Display or toggle command echo.

Related topics: Append, Echo, Log, Logging, Overwrite, Results

Echo [on | off]

on starts displaying entered Shell commands in the Transcript pane.

off stops displaying entered Shell commands in the Transcript pane.

With no argument, Echo displays its current setting.

You can also toggle echoing with the View menu Echo item.

EmuStatus

Report the current emulation status.

Related topics: $EMULATING, IsEmuHalted

EmuStatus

Use EmuStatus after IsEmuHalted returns no result.

```
isemuhalted;
emustatus;
// Processor is running.
halt;
// 961C60  0000 0000      ORI.B     #00,D0
isemuhalted;
// The emulator is halted.
```

The emulation status (halted or running) is also reported by the Status window or icon title and by the $EMULATING system variable.

EnableAlarmLimit

Enable a stack alarm limit.

Related topics: DisableAlarmLimit, DisableHighWaterMark, DisplayStack, EnableHighWaterMark, FillStackPattern, SetStackAlarm, SetStackArea, SetStackBase, SetStackSize, StackInfo

EnableAlarmLimit

If, when emulation halts, the stack usage is exceeding the alarm limit set by SetStackAlarm, you are notified.

You can also enable the alarm limit by checking the Stack window Options menu Enable Alarm Limit item.

EnableHighWaterMark

Track maximum stack usage.

Related topics:
DisableAlarmLimit, DisableHighWater-Mark, DisplayStack, EnableAlarmLimit, FillStackPattern, SetStackAlarm, SetStackArea, SetStackBase, SetStackSize, StackInfo

EnableHighWaterMark

This command enables an arrow on the Stack window stack meter to show the maximum stack area usage. The arrow moves when the stack grows to an address beyond any previously used. The arrow position is the stack high-water mark.

You can also enable the high-water mark by checking the Stack window Options menu Enable High-Water Mark item.

EventRestore

Retrieve saved event definitions.

Related topics:
EventSave

EventRestore "<filename>"

<filename> specifies a file containing event definitions. The quotation marks are required.

Events read from the file are added to the set of current events. Events from the file overwrite current events with the same name.

You can also restore events from a file with the Event window File menu Restore Events item.

EventSave

Save Events to a file.

Related topics:
EventRestore

EventSave "<filename>"

<filename> specifies the file in which to store current event definitions. The quotation marks are required.

You can also save events to a file with the Event window File menu Save Events item.

Exit

Exit the Shell window.

exit

This command closes the Shell window. To exit the emulator, open the Toolbar File menu and choose Exit. You can also close the Shell window with the Shell window File menu Exit item.

Fill

Fill memory with data.

Fill <addr1> <addr2> <data> [Byte | Word | Long | Dword] [<space>]

Related topics: Copy, Dump, RunAccess, Search, Size, Verify, Write

<addr1>	is the first address in the region to be filled. Addresses can be symbolic or numeric.
<addr2>	is the last address in the region to be filled.
<data>	is up to 256 bytes of data to be written. The value is repeated as needed to fill the region.
Byte	specifies the data is a byte value.
Word	specifies the data is a word value.
Long	specifies the data is a double word value.
Dword	is the same as Long.
<space>	specifies the emulator address space as:

- user, smm, or io for 386 EX emulators
- user or smm for 386 CX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

The physical write uses the Size command settings rather than the format size specified in the Fill command. For example, if Size=Byte, Fill uses byte-sized memory accesses.

Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Use RunAccess to enable Fill during emulation; however, such access can degrade your program execution.

```
Fill 0 1234 0x0 dword;   /* Fills memory from 0 to 64K with 0x0 */
// Fill successful.
```

You can also fill memory with the Memory window Edit menu Fill Memory item.

FillStackPattern

Initialize the stack.

FillStackPattern

Related topics: DisableAlarmLimit, DisableHighWaterMark,

With FillStackPattern, you can initialize the stack with a pattern to enable the stack usage statistics.

Other commands can also initialize the stack:

DisplayStack, EnableAlarmLimit, EnableHighWaterMark, SetStackAlarm, SetStackArea, SetStackBase, SetStackSize, StackInfo

- If you specify the stack base and size with FillStackArea, you can also initialize the stack in the single FillStackArea command.
- Enabling the high-water mark (the EnableHighWaterMark command) automatically fills the stack with the pattern.

Flush

Flush the Intel486 cache.

Flush

```
/* Disable cache so all code and data fetches appear on the bus */
Signal KEN disable                          /* Disable KEN# */
Flush                                       /* Flush the cache */
```

GDT

Display the global descriptor table.

Related topics: DT, IDT, LDT, PD, TSS

GDT (<selector> | <range> | <register>) [Base <address> [Limit <bytes>]] [All]

<selector> specifies a selector.

<range> specifies the first and last of a range of selectors.

<register> is any CPU mnemonic specifying a register containing a selector in the first 16 bits.

<address> specifies the descriptor table base address. The Base keyword is required.

<bytes> specifies a range of selectors as a number of bytes. The Limit keyword is required.

All displays all entries, including invalid or reserved.

With no arguments, GDT shows all valid entries in the range gdt_base to gdt_base+gdt_limit.

GDT displays the global descriptor table entries for a selector or range of selectors. The selectors displayed are determined by <selector>, <register>, Base <address> with either <range> or Limit <bytes>, or the current gdt_base and gdt_limit.

```
gdt 0x00 0x18 base 501010L;                 /* Display GDT entries*/
                          /* from 501018L (selector 0x08) to 501028L */
                        /* (selector 0x18). The table base is 501010L. */
```

GetBase

Get one or all base names and their address offsets.

Related topics: AddressOf, DisplaySymbols, NameOf, RemoveSymbols, SetBase

GetBase [<basename>]

<basename> displays only the specified base.

With no arguments, all bases loaded into the symbol table are displayed along with their offset values.

Compilers and linkers place symbols into groups called bases, assigning names to the groups. GetBase displays these symbol bases.

Go

Start emulation.

Related topics: $BREAKCAUSE, $EMULATING, Cause, GoInto, GoUntil, Halt, ResetAndGo, Step, StepSrc

Go

This command is equivalent to any of the following:

- Choose the Toolbar or Source window Go button.
- Choose the Source window Run menu Go item.
- Press the <F9> key.

GoInto

Emulate to a stepped-into or returned-into function.

Related topics: $BREAKCAUSE System Variable, $EMULATING System Variable, Cause, Go, GoUntil, Halt, ResetAndGo, Step, StepSrc

GoInto [Call | Return] [Line | Statement]

Call	If a call is executed within the current function, emulation continues through the call and into the called function, halting on the beginning of a line or statement. This line or statement can be the first instruction of the function or later, depending on how the compiler generates code and line-number start addresses.
Return	If a return is executed within the current function, emulation continues through the return, halting on the beginning of the next line or statement of the function returned to.
Line	breaks on a source line.
Statement	breaks on a source statement.

With no arguments specified, the first GoInto you use defaults to GoInto Call Statement. If you have previously used GoInto with arguments, any GoInto without arguments defaults to the arguments you used before.

You can also do these Go variations with the Source window buttons (configured by the Source window Options menu Set Go Buttons item) and from the Source window Run menu.

GoUntil

Emulate until a call or return.

Related topics: $BREAKCAUSE System Variable, $EMULATING System Variable, Cause, Go, GoInto, Halt, ResetAndGo, Step, StepSrc

GoUntil [Call | Return] [Line | Statement]

Call	within the current function, emulates until a call or return is executed.
Return	within the current function, emulates until a return instruction is executed.
Line	breaks on a source line.
Statement	breaks on a source statement.

With no arguments, the first GoUntil you use defaults to GoUntil Call Statement. If you have previously used GoUntil with arguments, any GoUntil without arguments defaults to the arguments you used before.

GoUntil emulates until a call or return is executed, then stops.

Because of how Call and Return work, the assembly instructions immediately before the call or return are not necessarily executed.

You can also do these Go variations with the Source window buttons (configured by the Source window Options menu Set Go Buttons item) and from the Source window Run menu.

Halt

Halt emulation.

Halt

Halt stops emulation when the current instruction finishes executing. This command is equivalent to any of the following:

- Choose the Toolbar or Source window Halt button.
- Choose the Source window Run menu Halt item.
- Press the <F2> key.

Help

Show Shell command syntax.

Help [<command>]

<command> is a Shell window command name.

Use Help to list, in the Transcript pane, the command syntax for one or more Shell window commands. With no argument, Help lists all commands alphabetically.

You can also pop-up on-line help from any SLD window Help menu or by pressing the <F1> key.

History

Control number of saved commands.

History [<size>]

<size> specifies the number of commands (0 to 50) to save in the Shell command history buffer.

With no arguments, History reports the current history buffer size.

Press <Ctrl><Up Arrow> or <Ctrl><Down Arrow> to recall commands sequentially from the history buffer to the Command Entry pane. You can edit recalled lines before entering them.

You can also set the history size with the Shell window Options menu History Size item.

IDT

Display the interrupt descriptor table.

IDT (<index> | <range> | <register>) [Base <address> [Limit <bytes>]] [All]

Related topics: DT, GDT, LDT, PD, TSS

<index> specifies an index.

<range> specifies the first and last of a range of selectors.

<register> is any CPU mnemonic specifying a register containing a selector in the first 16 bits..

<address> specifies the descriptor table base address. The Base keyword is required.

<bytes> specifies a range of indexes as a number of bytes. The Limit keyword is required.

All displays all entries, including invalid or reserved.

With no arguments, IDT shows all valid entries in the range idt_base to idt_base+idt_limit.

IDT displays the interrupt descriptor table entries for an index or range of indexes. The selectors displayed are determined by <index>, <register>, Base <address> with either <range> or Limit <bytes>, or the current idt_base and idt_limit.

```
idt 0x00 0x18 base 501010L                /* Display IDT entries */
                                  /* from 501018L (selector 0x08) to 501028L */
                                /* (selector 0x18). The table base is 501010L. */
```

If..Else

Conditionally execute Shell window commands.

If (<condition>) {<block>} [Else {<block2>}]

<condition>evaluates to nonzero or zero. The parentheses are required.

<block1> is a list of Shell commands, delimited with semicolons, to be executed when <condition> evaluates to nonzero. The braces are required.

<block2> is a list of Shell commands, delimited with semicolons, to be executed when <condition> evaluates to zero. The braces and Else keyword are required.

```
$a = 0;
If ($a) {
  "true";
}
else {
  "false";
};
// false
```

Include

Read commands from a file.

include "<filename>"

<filename> identifies a file containing Shell commands (a script). The quotation marks are required.

The commands are executed as if entered in the Command Entry pane. You can put an Include command in a script.

```
include "d:\shell.cmd";  /* executes d:\shell.cmd */
```

You can also run a script with the Shell window File menu Include item.

Integer

Identifies an integer.

Integer (<variable>)

<variable> is a Shell variable name. The parentheses are required.

Related topics:
String

Use Integer to discover whether a variable value is an integer. Integer returns 1 if <variable> is an integer and 0 otherwise.

```
$a = 0;
Integer($a);
// 1 1

If (integer($a)) { "it is an integer"; }
// it is an integer
```

IsEmuHalted

Discover whether emulator is halted.

Related topics: EmuStatus, $EMULATING

IsEmuHalted

Use IsEmuHalted to discover whether the emulator is halted. No response indicates the emulator is not halted. If you get no response, also use EmuStatus or $EMULATING.

```
isemuhalted;

halt;
// 961C60  0000 0000      ORI.B      #00,D0

isemuhalted;
// The emulator is halted.
```

The emulation status (halted or running) is also reported by the Status window or icon title and the $EMULATING system variable.

LapTimer

Takes a snapshot of the timer.

Related topics: StartTimer, StopTimer

LapTimer

Without stopping the timer, shows the number of milliseconds elapsed since the timer was started.

```
LapTimer;
while (laptimer < 5000) {};
```

LDT

Displays the local descriptor table.

Related topics: DT, GDT, IDT, PD, TSS

LDT (<selector> | <range> | <register>) [Base <address> [Limit <bytes>]] [All]

<selector> specifies the selector from the GDT to identify the LDT base and limit.

<range> specifies the first and last of a range of selectors.

<register> is any CPU mnemonic specifying a register containing a selector in the first 16 bits.

<address> specifies the descriptor table base address. The Base keyword is required.

<bytes> specifies a range of selectors as a number of bytes. The Limit keyword is required.

All displays all entries, including invalid or reserved.

With no arguments, LDT shows all valid entries in the range ldt_base to ldt_base+ldt_limit.

LDT displays the interrupt descriptor table entries for a selector or range of selectors. The selectors displayed are determined by <selector>, <register>, Base <address> with either <range> or Limit <bytes>, or the current ldt_base and ldt_limit.

```
ldt 0x00 0x18 base 501010L;          /* Displays LDT entries */
                        /* from 501018L (selector 0x08) to 501028L */
                    /* (selector 0x18). The table base is 501010L. */
```

Link

Establish source file links

Link [<file1> [<file2>]]

<file1> is a filename that has or needs a link.

<file2> is the filename to be linked to <file1>. Omitting <file2> displays the link already defined for <file1>.

With no arguments, Link displays all file links.

If the Source window fails to find <file1>, it searches for <file2>.

```
Link util.c util0215.c          // Use util0215.c wherever util.c
                                // is specified for source display.
```

List

List Shell variable values.

List [<variable>]

<variable> is a Shell variable name.

With no arguments, List displays all the Shell variables and their values.

```
List;
// (system) $SHELL_STATUS = 262158
```

Load

Load code and symbols to mapped or target memory.

Related topics:
LoadSize

Load "<filename>" [User | SMM] [[No]Code] [[No]Symbols] [[No]Demand] [[No]Demangle] [[No]UpdateBase] [Module <name>] [Reload] [[No]LoadRegister] [[No]Warn] [[No]Status]

<filename>	is the pathname of the file to be loaded. The quotation marks are required.
User	loads code into user memory.
SMM	for 386 EX, 386 CX, or Intel486 SLE emulators, loads code into system management mode memory.
[No]Code	loads or does not load code.
[No]Symbols	loads or does not load symbols.
[No]Demand	initially loads only global symbols (variables, module names, global function names, type definitions) and defers loading local symbolic information (local variables and line numbers) until needed or initially loads all symbols.
[No]Demangle	demangles or does not demangle C++ names.
[No]UpdateBase	updates symbol bases or does not update symbol bases, for OMF386 loadfiles on x86 emulators. Use updatebase in conjunction with loadregister.
<name>	after on-demand loading, loads symbols for the specified module Use this option in a script for debugging specific modules. Load symbols with this option to eliminate any delay on viewing a module. The Module keyword is required.
Reload	purges old symbols and loads new ones.
[No]LoadRegister	loads or does not load initial register values from OMF386 loadfiles.
[No]Warn	displays or does not display warnings from the loader.
[No]Status	displays or does not display load statistics.

With only <filename> specified, the default is Load "<filename>" User Code Symbols Demand NoDemangle NoUpdateBase NoLoadRegister NoWarn Status;

You can load code and symbols during emulation. Avoid loading into an area of memory occupied by the executing code. Loading into memory that is being executed can stop the emulator in an unpredictable

state.

```
Load demo.omf;
// 1986 bytes code loaded.
// 2 module(s) loaded.
// Load complete.
Load demo.omf module dm_main;     /* load code and symbols */
                                                /* from a module */
load demo.abs nocode;          /* load symbols only, on demand*/
Load demo.abs nosym;                          /* load code only */
load demo.abs nodemand;                    /* load all code; load */
                                          /*symbols on demand */
load sample.abs reload nowarn;    /* load code and symbols; */
                                          /* display no warnings*/
```

You can also load files with the Toolbar Load button or from the Source window File menu.

LoadSize

Set the memory write-access size for the load command.

Related topics: Load, Size

LoadSize [Byte | Word | Long | Dword]

Byte	writes memory by bytes.
Word	writes memory by words.
Long	(default) writes memory by longs. Writing in Long is the fastest way to load code.
Dword	is the same as Long.

Log

Display or set the name of the log file.

Related topics: Logging, Append, Overwrite, Echo, Results

Log ["<filename>"]

<filename> is the name of the logfile to be opened or created. The quotation marks are required.

With no arguments, Log displays the current log filename.

```
Logfile "c:\shell.log";
Log;
// log file name: c:\shell.log
```

You can also open a log file with the Options menu Log File Name item.

Logging

Display or toggle the logging setting.

Related topics: Log, Append, Overwrite, Echo, Results

Logging [on | off]

on starts echoing commands and results to the logfile.

off stops echoing commands and results to the logfile.

With no arguments, Logging reports whether logging is on.

In overwrite mode, each time you turn-on logging for a given logfile destroys prior information in that file. To preserve prior information, enter Append before Logging on.

You can also toggle logging with the Options menu Log Results item.

Map

Substitutes overlay memory for all or part of the target system memory.

Related topics: MapRanges, RestoreMap, SaveMap

Map [Clear | <base> [<end>] [Target] [<access>]] [<space>]

Clear clears all map blocks.

<base> is the address to start a memory region. The address is rounded down to the nearest boundary block equal to the amount of memory mapped:

- For PP-386 and SW-386 emulators, you can map any multiple of 4K bytes starting on any 4K address.
- For EA-486 emulators, you can map any multiple of 128K bytes starting on any 128K address.
- For EA-NS 486 emulators, you can map any multiple of 64K bytes starting on any 64K address.

<end> is the last address of the region. This address is rounded up to the top of the region containing the end address, as described for the <base> argument above.

Target maps the memory region to target memory.

<access> specifies access permissions. Your emulator offers some or all of the following ways to control and report your program's memory accesses:

RAM allows read and write access (the default).

ROM allows read access; prevents write access; does not break on attempted write access. (Intelx86 emulators allow writes to target memory.)

ROMbrk allows read access; prevents write access; breaks on attempted write access. (Intelx86 emulators

allow writes to target memory but such writes break emulation.) This option is unavailable on NS486 emulators.

None prevents any access; breaks on attempted access. (Intelx86 emulators allow access to target memory but such access breaks emulation.) This option is unavailable on NS486 emulators.

<space> specifies the emulator address space as:

- user, smm, or io for 386 EX emulators
- user or smm for 386 CX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

With no arguments, Map displays the current map settings.

```
map 0 ram;
// Mapped block starting at address 00000000 to 0000FFFF RAM
```

You can also map memory with the Toolbar Map button.

MaxBitFieldSize

Set the maximum bit field size for OMF386 loadfiles.

MaxBitFieldSize [16 | 32]

16 Sets the maximum bit field size to 16 bits for Borland C compiler-generated OMF386 loadfiles.

32 Sets the maximum bit field size to 32 bits (default) for all other loadfiles.

NameOf

Find the symbol representing an address.

Related topics: AddressOf, DisplaySymbols, GetBase, RemoveSymbols, SetBase

NameOf <address>

<address> is a numeric address.

Use NameOf to look up a specified address and display the symbol that most closely matches the address.

```
NameOf 0x0900;
// #main#14#1 (function main)
```

Overwrite

Overwrites the log file.

Related topics:
Append, Echo, Log, Logging, Results

Overwrite

When Overwrite has been specified, starting to log (Logging On) destroys any prior logfile contents.

You can also configure logging to overwrite prior information with the Shell window Options menu Overwrite Log File item.

PD

Display the page directory.

Related topics:
DT, GDT, IDT, LDT, TSS

PD [range]

<range> is the address range of the entries to be displayed.

With no argument, PD displays the first eight page directory entries.

```
reg cr3 0x5e0000;
write 0x5e0000p 0x12345007 0x56789067 0x0 0x0 0x0 0x0 0x0 dword;
// Write successful.
pd;                                          // same as pd 0x0L 0x02000000L;
// 00000000L present                    user read/write table=12345000
// 00400000L present accessed dirty user read/write table=56789000
// 00800000L NOT PRESENT
// 00C00000L NOT PRESENT
// 01000000L NOT PRESENT
// 01400000L NOT PRESENT
// 01800000L NOT PRESENT
// 01C00000L present      dirty supervisor read/write table=6F04C000
//
```

Pmode

Displays the processor mode.

Pmode

The x86 processors operate in various address modes (pmodes). These are real, virtual-86 (V86), protected, and System Management Mode (SMM). Protected mode is further divided into 16-bit and 32-bit protected modes.

The Intel386 DX and Intel386 SX processors have no SMM.

```
pmode;
// Processor mode = Prot32
```

The pmode also appears at the bottom of the Status window icon.

Print

Display a Shell variable or string constant value.

Print (<variable> | "<string>")

<variable> is the name of a Shell variable.

<string> is a string constant. The quotation marks are required.

The parentheses are required.

```
$a = 5;
Print ("abc");
// abc
Print($a);
// 0x5 5
```

RAMtst

Run the memory hardware confidence tests.

Related topics:
Test

RAMtst [Loop] <address1> <address2> [<space>]

Loop repeats the low-level operations in the specified test so the operation can be observed on an oscilloscope. Press <Esc> to stop looping. An error does not halt the test loop.

<address1>is the first address in the range to test.

<address2>is the last address in the range to test.

<space> specifies the emulator address space as:

- user, smm, or io for 386 EX emulators
- user or smm for 386 CX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

The tests appropriate for your emulator are described in the *Hardware Reference*.

```
ramtst 0x0000 0xFFFF;      /* Test memory from 0x0 to 0xffff. */
```

Register

Display or set register values.

Register [<name> [<value>]] [...]

<name> is a CPU register mnemonic.

<value> is the value to be put into the register.

With no arguments, Register displays all the registers. A <name>

without a <value> displays the value of the specified register.

You can also view and edit the registers in the CPU window.

RemoveSymbols

Remove all loaded symbols and clear all allocated symbol tables.

Related topics:
AddressOf, DisplaySymbols, GetBase, Load, NameOf, SetBase

RemoveSymbols

Reset

Reset the target or processor.

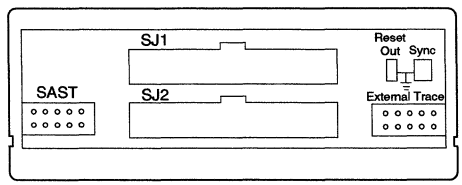
Related topics:
ResetAndGo

Reset [CPUonly | Target]

CPUonly resets the processor without updating the SLD windows. Use this argument only if **Reset** without **CPUonly** fails to reset the processor:

1. Enter **Reset CPUonly**, resetting the processor without updating the SLD windows.
2. Reset your target.
3. Enter **Reset** again, without **CPUonly**, to update the SLD windows.

Target puts a pulse signal on the SW or EA emulator Reset Out pins for approximately one millisecond. For Reset Out signal values, see **[SystemInfo]** in the "powerpak.ini File Reference" chapter. The Reset Out pins are on the front panel of the SW or EA emulator, as shown in the following figure. The bottom pin is grounded. Connect these pins to a reset or other appropriate input on your target board.



With no argument, **Reset** sends a RESET signal to the processor. All

CPU register contents are lost on reset:

- The processor RESET pin is asserted.
- The program counter and stack pointer are reset and other segment registers are set to 0. The Source window displays the program counter location. The Stack window display becomes invalid.
- All SLD windows are updated.

You can also reset the emulator from the Toolbar Configure menu, the Source window Run menu, or the CPU window Options menu.

ResetAndGo

Assert and release the target reset line.

Related topics:
Reset

ResetAndGo

This operation is required to start some target systems. For example, targets that use an external watchdog timer or power-saver hardware may require that you use ResetAndGo.

You can also reset the processor and start emulation with the Source window Run menu Reset And Go item.

ResetLoaders

Reinitialize the loaders when you get an error message telling you to do so.

ResetLoaders "<loadpath>"

<loadpath> is the path to the directory containing loaders.ini. The quotation marks are required.

With no argument, ResetLoaders uses the SLD directory.

RestoreCS

Restores the chip-select register values.

Related topics:
SaveCS

RestoreCS "<filename>"

<filename> is an ASCII file describing chip select register values. The quotation marks are required.

The ASCII file contains an entry for each register. Each entry can be up to 80 characters long, containing the following sequential fields:

<REGISTER NAME in upper case>
<1 to 20 spaces>
<hexadecimal value>
<new line or white space>
<any comment text other than 0A or 0>

You can create the chip select file with a SaveCS command. For a processor-specific list of registers, see the *Hardware Reference*.

You can also restore the chip selects with the Toolbar Configure menu Restore Chip Selects item.

RestoreMap

Restores a saved map configuration.

Related topics:
Map, MapRanges, SaveMap

RestoreMap "<filename>"

<filename> is a file containing a map configuration. The quotation marks are required.

You can also restore the map from a file with Map dialog box Restore button, accessible via the Toolbar Map button.

Results

Set the Transcript pane results display.

Related topics:
Append, Echo, Log, Logging, Overwrite, Results

Results [on | off]

on (default) displays Shell command results in the Transcript pane.

off displays no Shell command results in the Transcript pane.

Without arguments, Results displays the current setting.

You can also toggle results with the View menu Show Results item.

RunAccess

Set the target processor access mode during emulation.

Related topics:
Copy, Dump, Fill, Search, Size, Verify, Write

RunAccess [on | off]

off (default) disables reading and writing memory during emulation.

on enables reading and writing memory during emulation.

Without arguments, RunAccess displays the current setting.

Memory access is used for operations that read and write the peripheral registers and memory, including scrolling or updating the Peripheral and Memory window displays. Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Use RunAccess to enable memory accesses during emulation; however, such access can degrade your program execution.

You can also toggle Run Access with the Toolbar Configure menu Run Access item.

SaveCS

Saves the chip-select registers.

Related topics: RestoreCS, ConfigCS

SaveCS "<filename>"

<filename> is the filename where the chip select register values are to be saved. The quotation marks are required.

Different chip select registers are saved for different processors. See the processor-specific lists in the *Hardware Reference*.

Restore the register values from the file with RestoreCS.

You can also save the chip selects with the Toolbar Configure menu Save Chip Selects item.

SaveMap

Saves a memory map configuration.

Related topics: RestoreMap

SaveMap "<filename>"

<filename> specifies the pathname of the file where the memory map is to be saved. The quotation marks are required.

Restore the map from the file with RestoreMap.

You can also save the map from the Map dialog box, accessible from the Toolbar Map button.

Search

Search for a pattern in memory.

Related topics: Copy, Dump, Fill, RunAccess, Size, Verify, Write

Search <start> <end> [Not] <data> [Byte | Word | Long | Dword] [<space>]

<start>	is the first address in the address range to be searched. Addresses can be symbolic or numeric.
<end>	is the last address in the range to search.
Not	searches for the first pattern mismatch rather than the first pattern match.
<data>	is a pattern for which to search, up to 256 bytes long.
Byte	specifies the data is a byte value.
Word	specifies the data is a word value.
Long	specifies the data is a double word value.
Dword	is the same as Long.
<space>	specifies the emulator address space as:

- user, smm, or io for 386 EX emulators

- user or smm for 386vCX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

The physical read of memory uses the Size command settings rather than the format size set by the Search command. For example, if Size=Byte, Search reads memory in byte-sized memory accesses.

Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Use RunAccess to enable Search during emulation; however, such access can degrade your program execution.

```
Fill 0 ffff 0x0 user;
Write 400 0x1234 user;

Search 0 ffff 0x1234 user;
// pattern found at 400
```

You can also search for a pattern in memory with the Memory window Edit menu Search Memory item.

SetBase

Relocate symbols.

Related topics: AddressOf, DisplaySymbols, GetBase, NameOf, RemoveSymbols

SetBase <base> <address>

<base> is the base name for the symbols to be relocated. Case is significant.

<address> is the new numeric or symbolic base address.

SetBase relocates the symbols in the specified <base> to their offset addresses plus the specified <address>. The default base address is 0.

You can use SetBase to quickly relocate all symbols in a base. For example, if code is loaded by the target program into memory other than where it was linked, you can set the base address to the new load address using SetBase, thus matching the code symbol addresses to the memory where the code is loaded.

To discover the base names and their address offsets, use GetBase.

SetStackAlarm

Set the stack alarm limit.

Related topics:
DisableAlarmLimit, DisableHighWater-Mark, DisplayStack, EnableAlarmLimit, EnableHighWater-Mark, FillStackPattern, SetStackArea, SetStackBase, SetStackSize, StackInfo

SetStackAlarm <percent>

<percent> is a percentage of the stack area, from 1 to 99.

Use SetStackAlarm to set the stack alarm limit as a percentage of the stack. The alarm appears as a red line on the stack meter in the Stack window.

When enabled (see EnableAlarmLimit), the stack alarm notifies you if the stack usage is exceeding the alarm limit when emulation halts.

You can also set the stack alarm with the Stack window Options menu Alarm Limit item.

SetStackArea

Redefine the stack location and size.

Related topics:
DisableAlarmLimit, DisableHighWater-Mark, DisplayStack, EnableAlarmLimit, EnableHighWater-Mark, FillStackPattern, SetStackAlarm, SetStackBase, SetStackSize, StackInfo

SetStackArea [<address> <size> [fillArea]]

<address> is a numeric or symbolic base address.

<size> is the stack size in bytes.

fillArea initializes the stack area.

With no arguments, SetStackArea shows the current settings (the same as StackInfo).

This command changes the addresses used by the emulator for stack monitoring and does not affect your program's stack allocation.

Separate Shell commands (SetStackBase and SetStackSize) exist to set the stack base and size. Because of the delay between command executions, using the separate commands to redefine the stack can temporarily define an invalid stack area for the emulator's stack monitoring operations. SetStackArea sets the stack base and size in a single command.

To fill the stack area with a pattern without changing the base and size, use FillStackPattern.

```
setstackarea 0x1000 0x500 fillarea;
```

You can also set the stack base and size with the Stack window Options menu Stack Area item.

SetStackBase

Set the stack base address.

Related topics:
DisableAlarmLimit,
DisableHighWater-Mark,
DisplayStack,
EnableAlarmLimit,
EnableHighWater-Mark,
FillStackPattern,
SetStackAlarm,
SetStackArea,
SetStackSize,
StackInfo

SetStackBase <address>

<address> is a numeric or symbolic base address.

This command changes the base address used by the emulator for stack monitoring and does not affect your program's stack base address.

Separate Shell commands exist to set the stack base and size. Because of the delay between command executions, using separate commands to redefine the stack can temporarily define an invalid stack area for the emulator's stack monitoring operations. SetStackArea sets the stack base and size with a single command.

To show the current stack settings, use StackInfo.

```
SetStackBase F000;
```

You can also set the stack base with the Stack window Options menu Stack Area item.

SetStackSize

Set the stack size.

Related topics:
DisableAlarmLimit,
DisableHighWater-Mark,
DisplayStack,
EnableAlarmLimit,
EnableHighWater-Mark,
FillStackPattern,
SetStackAlarm,
SetStackArea,
SetStackBase,
StackInfo

SetStackSize <size>

<size> is the stack size in bytes.

This command changes the stack size used by the emulator for stack monitoring and does not affect your program's stack size.

Separate Shell commands exist to set the stack base and size. Because of the delay between command executions, using separate commands to redefine the stack can temporarily define an invalid stack area for the emulator's stack monitoring operations. SetStackArea sets the stack base and size with a single command.

To show the current stack settings, use StackInfo.

```
SetStackSize 200;
```

You can also set the stack size with the Stack window Options menu Stack Area item.

Signal

Display or set whether signals are enabled.

Signal [[<name> [Enable | Disable]] | [All Enable | All Disable]]

Enable drives the specified signal by your target system.

Disable	drives the specified signal by the emulator.
All Enable	connects all signals.
All Disable	disconnects all signals.
<name>	identifies a signal. For a processor-specific list of configurable signals, see the *Hardware Reference*.

With no arguments are specified, Signal displays the status of all signals. To display the status of a particular signal, specify only <name>.

```
signal;
// READY# ENABLE
// RESET  DISABLE
// HOLD DISABLE
// NMI DISABLE
// NA# DISABLE
// INTR DISABLE
// Coprocess DISABLE
signal reset enable;
// RESET  ENABLE
```

You can also toggle the signal connections with the CPU window Options menu Signals item.

Size

Selects memory access size.

Related topics: Copy, Dump, Fill, RunAccess, Search, Verify, Write

Size [Byte | Word | Long | Dword]

Byte, Word, Long, and Dword specify the size of subsequent memory accesses. Dword is the same as Long. The memory access size is independent of the display size.

With no argument, Size reports the current setting.

You can also specify the memory access size from the Memory window Options menu.

StackInfo

Display the stack information.

Related topics: DisableAlarmLimit, DisableHighWaterMark,

StackInfo

This command displays the current stack information. The number of frames shows the call nesting level.

```
StackInfo;
// stack base = 12345678
```

DisplayStack, EnableAlarmLimit, EnableHighWater-Mark, FillStackPattern, SetStackAlarm, SetStackArea, SetStackBase, SetStackSize

```
// size = 0
// current stack pointer = 87654321
// frames = 0
// alarm limit = 0%, DISABLED
// high water mark = 00000000
// stack type = high to low
```

The same information appears in the Stack window.

StartTimer

Start the timer.

Related topics: LapTimer, StopTimer

StartTimer

This command resets the elapsed time to zero and starts the timer.

Step

Emulate one or more instructions.

Related topics: $BREAKCAUSE System Variable, $EMULATING System Variable, Cause, Go, GoInto, GoUntil, Halt, ResetAndGo, StepSrc

Step [Into | Over] [<count>]

Into	(default) if a function call is encountered, executes the function call as a step and continues according to <count> within the called function.
Over	if a function call is encountered, executes the entire function (including any functions it calls) as a single step and continues according to <count> within the calling function.
<count>	specifies how many steps to do. A large <count> can cause stepping to go for a long time. Press <ESC> to break out of stepping before the step count is finished.

The default granularity and count are determined by the Source window Options menu Source Step Granularity and Step Count items.

You can also step with the Toolbar Step button, various Source window buttons, and the Source window Run menu.

StepSrc

Step emulation by source lines or statements.

Related topics: $BREAKCAUSE

StepSrc [Into | Over] [Line | Statement] [<count>]

Into	(default) if a function call is encountered, executes the function call as a step and continues according to <count> within the called function.

System Variable, $EMULATING
System Variable, Cause, Go, GoInto, GoUntil, Halt, ResetAndGo, Step

Over	if a function call is encountered, executes the entire function (including any functions it calls) as a single step and continues according to <count> within the calling function.
Line	sets the step granularity as one source line. A source line can contain more than one statement. Lines can be out-of-order relative to the sequence of instructions the compiler generates.
Statement	sets the step granularity as one statement.
<count>	specifies how many steps to do. A large <count> can cause stepping to go for a long time. Press <ESC> to break out of stepping before the step count is finished.

The default granularity and count are determined by the Source window Options menu Source Step Granularity and Step Count items.

You can also step with the Toolbar Step button, various Source window buttons, and the Source window Run menu.

StopTimer

Stop and report on the timer.

Related topics: LapTimer, StartTimer

StopTimer

Stop the timer and show the number of milliseconds elapsed since the previous StartTimer command.

String

Discover whether a variable is a string.

Related topics: Integer

String (<variable>)

<variable> is a Shell variable name. The parentheses are required.

String returns 1 if the variable is a string and 0 otherwise.

```
$a = "qrs";
String($a);
// 0x1 1
if (string($a)) { "it is a string"; }
// it is a string
```

SymbolCloseFile

Close the symbol text file.

DisplaySymbols, SymbolOpenFile

SymbolCloseFile

Closes the file opened by SymbolOpenFile.

SymbolOpenFile

Open a text file.

DisplaySymbols, SymbolCloseFile

SymbolOpenFile "<filename>"

<filename> is the name of a file. The quotation marks are required.

Opens a text file with the specified filename. Subsequent output from DisplaySymbols is directed to the specified file. The file can be viewed with an editor or file browser.

Test

Run the hardware confidence tests.

Related topics: Ramtst

Test [Loop] [Repeat | Continue] [Brief | Verbose] [Except] [<name> | <number>]

Loop	repeats the low-level operations in the specified test so the operation can be observed on an oscilloscope. Press <Esc> to stop looping.
Repeat	repeats the specified test until you press <Esc>.
Continue	continues through all tests, even if one fails.
Brief	displays only the final test result.
Verbose	displays every test result and progress report.
Except	excludes the specified tests and runs all others.
<name>	specifies one or more tests by name.
<number>	specifies one or more tests by number.

With no arguments, Test runs all tests and displays the results.

To run these tests, connect the Stand-Alone Self-Test (SAST) or null target board as described in the *Hardware Reference*. The tests appropriate to your emulator are also described in the *Hardware Reference*.

Time

Show the current date and time.

Time

Transcript

Set the number of lines saved in the transcript pane.

Related topics:
Echo, Results

Transcript [<size>]

<size> is the number (0 to 1000) of lines to be saved in the scrollable Transcript pane.

You can also set the transcript size with the Options menu Set Transcript Size item.

TSS

Displays task state segments.

Related topics:
DT, GDT, IDT, LDT, PD

TSS (<selector> | <register>) [Base <address> [Limit <bytes>]] [Tss286 | Tss386]] [All]

<selector> specifies the selector from the GDT to identify the TSS base and limit. With no selector specified, the current tss_base and tss_limit are used.

<register> is any CPU mnemonic specifying a register containing a selector in the first 16 bits.

<address> specifies the descriptor table base address. The Base keyword is requried.

<bytes> specifies a range of selectors as a number of bytes. The Limit keyword is required.

All Displays all entries, including invalid or reserved entries.

Tss286 specifies Intel286 processor segmentation.

Tss386 specifies Intel386 processor segmentation.

With no entries, TSS displays all task state segments plus the I/O bit map in the range tss_base to tss_limit.

TSS displays the task state segments for any selector or base address.

Verify

Toggles on and off a read-after-write.

Verify [on | off]

on checks values written to memory (default).

Related topics:
Copy, Dump, Fill, Load, RunAccess, Search, Size, Write

off does not check writes.

Verify checks writes by reading-back the written value and comparing the read value with the value supposedly written. If they do not match, an error is returned. Verification can happen after a Write, Fill, or Load. Verification does not affect the target processor during emulation.

You can also toggle write verification with the Memory window Options menu Write Verify item.

Version

Report the emulator version information.

Version

Use version when logging an emulator session to record which version of the emulator hardware, software, and firmware is in use. The information from this command is also needed when you contact Microtek for technical support or product upgrades.

You can also view some version information from the Toolbar Help menu About item.

While

Repeatedly execute statements while the condition is true.

Related Topics:
If...Else

While (<condition>) { <block> }

<condition> evaluates to true (non-zero) or false (zero). The parentheses are required.

<block> is one or more Shell commands delimited with semicolons. The braces are required.

While <condition> is true, the <block> executes.

```
$a = 0;  While ($a < 500) {$a = $a + 1;}
```

Write

Write to a memory address.

Related topics:
Copy, Dump, Fill, RunAccess, Search, Size, Verify

Write [Loop] <address> <data> [Byte | Word | Long | Dword] [<space>]

Loop repeats the write without printing, even if errors occur.

<address> is a numeric or symbolic starting address.

<data> is up to 256 data values to be written.

Byte	specifies the data is a byte value.
Word	specifies the data is a word value.
Long	specifies the data is a double word value.
Dword	is the same as Long.
<space>	specifies the emulator address space as:

- user, smm, or io for 386 EX emulators
- user or smm for 386 CX and Intel486 SLE emulators
- user or io for NS486 emulators
- user for 386 SX and Intel486 non-SLE emulators

The physical write to memory uses the Size command settings rather than the format size specified in the Write command. For example, if Size=Byte, Write commands write by byte-sized memory accesses.

Because reading and writing memory takes a small amount of processor time, memory access is initially disabled during emulation. Use RunAccess to enable Write during emulation; however, such access can degrade your program execution.

Xlt

Translates an x86 numeric address.

Related topics:
AddressOf,
NameOf

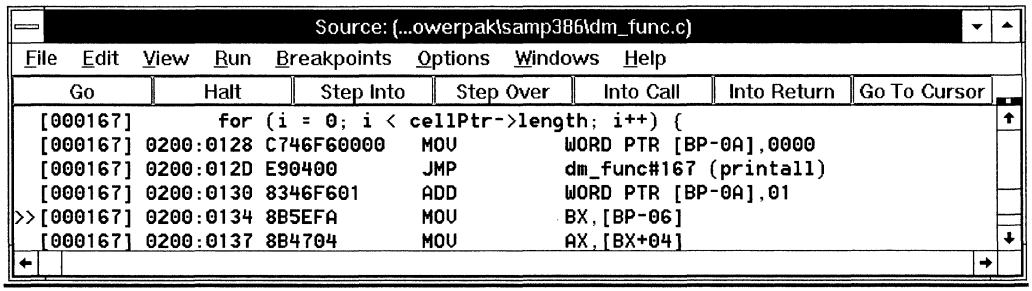
Xlt <address>

<address> is a numeric or symbolic address.

Xlt translates any numeric or symbolic address to its equivalent linear or physical form, according to x86 numeric addressing rules. For a virtual address, Xlt displays the linear and physical equivalents. For linear or physical addresses, Xlt displays the physical equivalent.

```
Xlt #upper#startup
// 0020:00F35BD0 = 00F35BD0L = F35BD0P
```

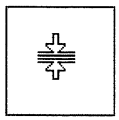
Source Window Reference



Source Window Contents

The Source window displays:

- when enabled, the source line numbers
- when available, the source lines
- when enabled, the disassembly corresponding to each source line, including the load address, hexadecimal code, and instructions



You can display two independently scrolling Source window panes. To reveal the second pane, drag the split box cursor (see figure at left) above the top arrow of the vertical scroll bar. To change focus to a pane, click in the inactive pane or press <Tab>.

Source Window Menus

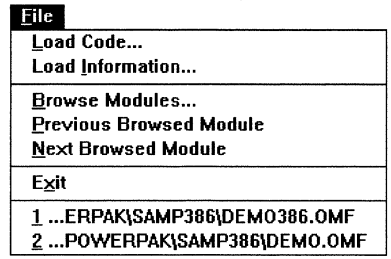
Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	**Use To:**
File	Load and view modules; close the Source window.
Edit	Navigate through source.
View	Configure the source and disassembly display.
Run	Start or stop emulation; step; reset.
Breakpoint	Define and manage breakpoints.
Options	Manage source display options and emulation controls.

Windows	Open another SLD window.
Help	Open a window for help on the SLD software.

File Menu

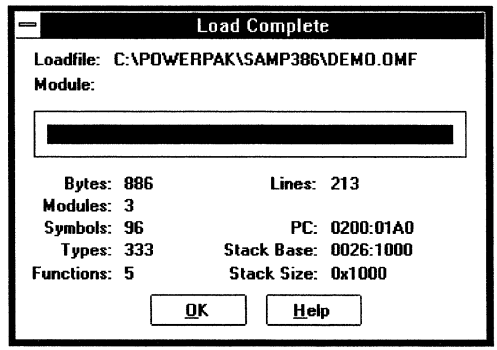
File menu, showing previously loaded files demo386.omf and demo.omf



Load Code... opens the Load dialog box to load code or symbols from a loadfile. This has the same effect as choosing the Toolbar Load button, as described in the "Toolbar Reference" chapter. To reload a file, choose from the (up to four) files listed at the bottom of the Source window File menu.

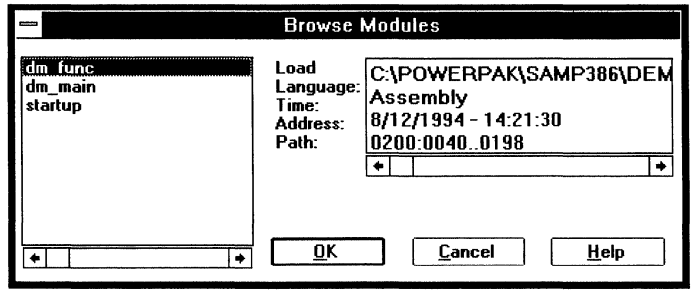
Load Information... opens an information box describing the loadfile and what has been loaded into the emulator. The display differs between processors.

Load Information dialog box, describing a successful demo.omf load



Browse Modules... opens a dialog box to change the module (source, disassembly, and symbols) displayed in the Source window.

Browse Modules dialog box, with dm_func selected



To select a module, click on the module name or use the <Up Arrow> and <Down Arrow> keys to scroll the cursor. For the selected module, the dialog box displays:

Load File:	The loadfile path and filename
Language:	The language of the source file
Time:	The date and time the loadfile was created
Address:	Where in memory the module is loaded
Path:	The source file path and filename

Choose OK to browse to the selected module or Cancel to exit the dialog box without changing the Source window display.

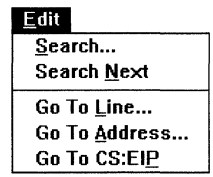
Previous Browsed Module changes the Source window display back to the module you last viewed. The SLD software maintains a list of which modules you have browsed and in what order you browsed them.

Next Browsed Module changes the Source window display to the next module in the browse history.

Exit closes the Source window. To exit the SLD software, use the Toolbar File menu Exit item.

1, **2**, **3**, **4** lists the last four files you loaded. Reload a file by choosing it from this list. This method of reloading a file bypasses the Load and Load Options dialog boxes.

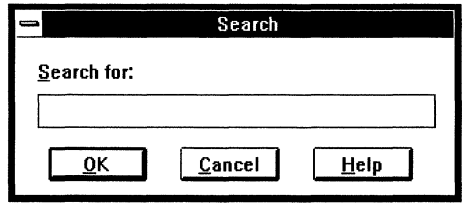
Edit Menu



Search opens a dialog box for searching the Source window text for a specific string. Case is significant in the search string. The search

starts from the Source cursor and stops at the first instance of the string found. If the string is not found, the search stops at the end of the module. To search the entire module, position the Source cursor at the beginning of the module before starting the search.

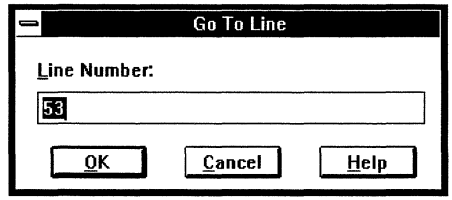
Search dialog box, for finding a string in the source display



Search Next searches again for the last string you entered in the Search dialog box. The search starts from the cursor and stops at the first match or the end of the module.

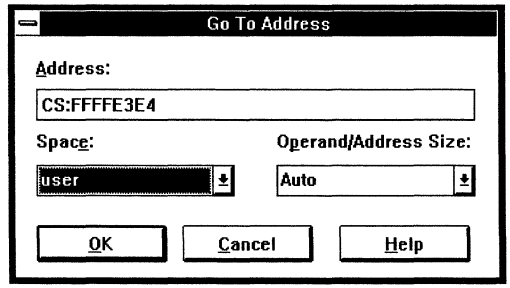
Go To Line... opens a dialog box to move the Source cursor to a specific line. If you specify a line number beyond the last line in the current module, the Source cursor moves to the end of the module.

Go To Line dialog box, for finding a line number in the source display



Go To Address... opens a dialog box to move the Source cursor to a specific address. If no source is available for the address you specify, the Source window shows disassembled code beginning at that address.

Go To Address dialog box, for finding code load address FFFFE3E4 in User space



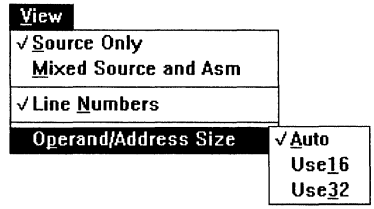
You can specify:

Space:	User or SMM (system management mode), depending on the processor
Operand/Address Size:	Use16 (16-bit), Use32 (32-bit), or Auto (pmode-derived) addressing mode.

Go To CS:EIP moves the Source cursor to the current program counter.

View Menu

View menu, showing the Operand/Address Size sub-menu



Source Only, when checked, displays only your source code.

Mixed Source and Asm, when checked, displays the source code lines interleaved with the corresponding disassembly lines from memory.

Line Numbers, when checked, displays the source file line numbers

Operand/Address Size opens a sub-menu with the following choices to display disassembly text:

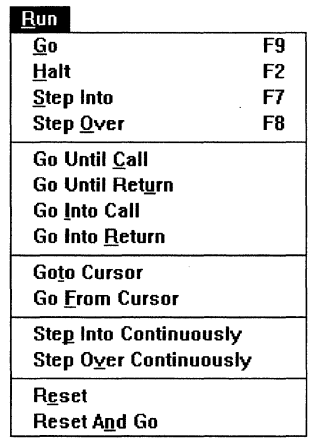
Auto Operand/address size is 16-bit or 32-bit, depending on the pmode.

Use16 Operand/address size is 16-bit.

Use32 Operand/address size is 32-bit.

Run Menu

Run menu, listing the keyboard-shortcut function keys for emulation control



Go or pressing <F9> starts emulation.

Halt or pressing <F2> stops emulation.

Step Into or pressing <F7>, when the program counter is on a function call, executes the call to the function and stops before the first instruction in the function. Step Into and Step Over are the same operation when the program counter is not on a function call.

To step into a function with no associated source, before stepping enable the View menu Mixed Source and Asm item to display disassembly in the Source window. Otherwise, Step Into operates the same as Step Over for that function. The Source window must be able to display the program counter where emulation halts.

Step Over or pressing <F8>, when the program counter is on a function call, executes the call as a single step. This step executes the function, returns, and stops before the first instruction following the return. (However, encountering a breakpoint in the stepped-over function stops emulation at the breakpoint.) The Source window continues to display the calling function.

Go Until Call executes from the program counter to the beginning of a statement or line (depending on the granularity) containing a call.

Go Until Return executes from the program counter to the beginning of a statement or line (depending on the granularity) containing a return.

Go Into Call executes from the program counter and stops before the first instruction in the next called function.

Go Into Return executes from the program counter through the first return instruction and stops before the first instruction after the return.

Go To Cursor executes from the program counter and stops before the selected line or statement in the Source window.

Go From Cursor moves the program counter to the selected line or statement in the Source window, then starts emulation.

Step Into Continuously does Step Into operations until you halt it.

Step Over Continuously does Step Over operations until you halt it.

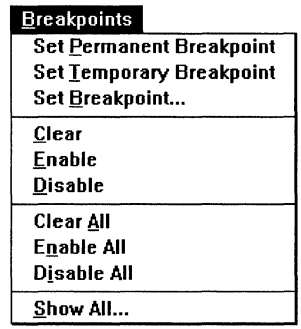
Reset asserts the RESET pin of the target processor, causing the CPU to reset the internal registers, the program counter, and the stack pointer. The RESET pin is then released. All SLD windows are updated; the Source window displays the beginning of code (where the program counter points) and the Stack window display is invalid.

Reset And Go does a Reset, as above, and starts emulation from the power-up reset vectors. The reset vectors must be previously set.

Breakpoints Menu

Set Permanent Breakpoint, Set Temporary Breakpoint, Set Breakpoint..., and Show All... are always available; Clear, Enable, and Disable are available when you have selected a breakpoint from those listed in the Breakpoint window; Clear All, Enable All, and Disable All are available when one or more breakpoints are listed. To select a breakpoint, click on it or move the highlight with <Up Arrow> and <Down Arrow> keys.

Breakpoints menu with all items enabled, indicating at least one breakpoint is defined

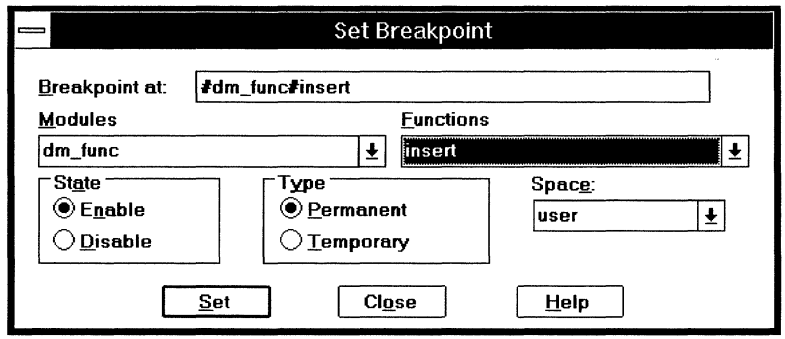


Set Permanent Breakpoint sets a permanent breakpoint at the cursor.

Set Temporary Breakpoint sets a temporary breakpoint at the cursor.

Set Breakpoint... opens a dialog box to set a breakpoint at a specific address.

Set Breakpoint dialog box to set a permanent, initially enabled breakpoint at address 73 (hexadecimal) in the dm_main module main function



Fill-in the dialog box as follows:

Breakpoint at: can be a numeric or symbolic address. For symbolic addresses, choose a module and a function from the drop-down list boxes.

State can be toggled to Enable or Disable. The emulator ignores a disabled breakpoint.

Type	can be permanent or temporary. A temporary breakpoint is removed after it causes the break.
Space:	can be User or SMM for some processors.

Choose the Set button to define the breakpoint or the Close button to close the dialog box without defining a new breakpoint.

Clear removes a breakpoint at the Source cursor.

Disable marks the breakpoint at the Source cursor to be ignored when emulation executes through the code where the breakpoint is located. A disabled breakpoint highlight in the Source window is grey.

Enable marks the breakpoint at the Source cursor to cause a break when emulation executes through the code where the breakpoint is located. An enabled breakpoint highlight in the Source window is red.

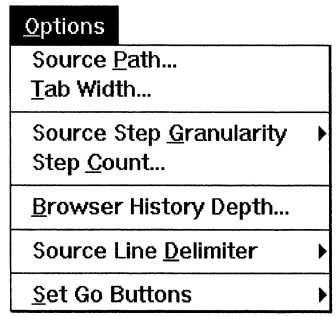
Disable All disables all currently defined breakpoints. The breakpoints remain defined.

Enable All enables all currently defined breakpoints.

Clear All removes all breakpoints. No breakpoints remain defined.

Show All... opens the Breakpoint window, described in the Breakpoint Window Reference chapter.

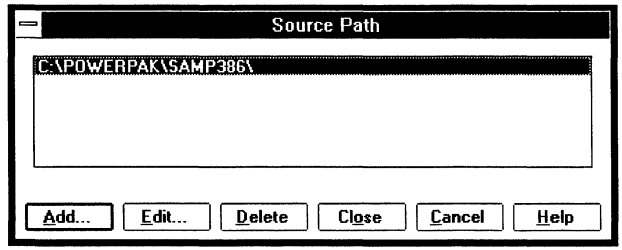
Options Menu



Source Path opens a dialog box to add, delete, or change the paths to the source files used in generating your loadfile. You can define up to 50 source paths. The path list is saved in powerpak.ini.

When you browse a module in the Source window, the emulator searches the source paths for the corresponding source file in the order they appear in the dialog box, from top to bottom.

Source Path dialog box, specifying c:\powerpak\samp386 as the path for all source files



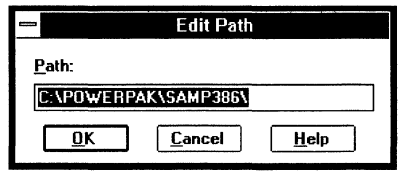
To select a source path for editing or deleting, click on it or use the <Up Arrow> and <Down Arrow> keys to move the highlight.

The Source Path dialog box buttons are:

Add... opens a dialog box for adding a new source path to the emulator's list of source paths. Select a source file; choose OK to add the path or Cancel to close the dialog box without adding the path.

Edit... opens a dialog box for editing the selection.

Edit Path dialog box for changing the c:\powerpak\samp386 entry in the Source Path dialog box



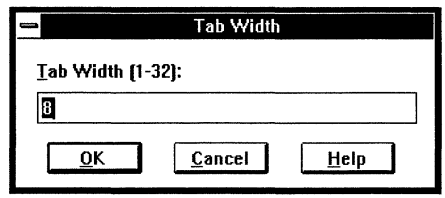
Delete removes the selection from the source path list.

Cancel closes the Source Path dialog box, first asking you whether to keep or abandon the Add, Edit, and Delete changes.

Close replaces Cancel when you click on OK. This button closes the Source Path dialog box, keeping all changes you have made.

Tab Width... opens a dialog box to specify the number of spaces the Source window uses to replace a tab character in your source file. The default is eight spaces.

Tab Width dialog box, replacing tabs with 8 spaces



Tab Width And Statement-Level Breakpoints

To set a breakpoint at the statement level, you must know how many spaces your compiler uses for a tab character. For example:

```
<tab><tab>for( i = 0; i < MAX_NUM; I++){          /*source line*/
```

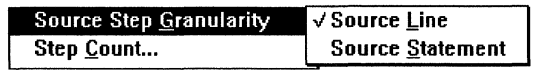
The compiler generates column range information for the three statements in this line, using a tab width of 8:

i = 0	columns 0 to 26
i < MAX_NUM	columns 27 to 39
i++	columns 40 to 45

If you set the Source window Tab Width to 4, then use the Source cursor to set a breakpoint on the first i (column 13) or the second i (column 20), the breakpoint is within the first statement's column range. The third i is within the second statement's range.

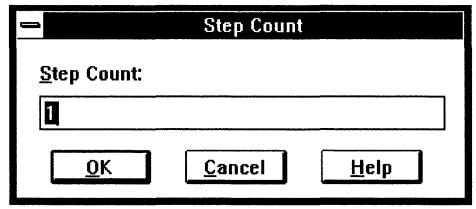
Source Step Granularity opens a sub-menu to specify whether a Step command steps by source lines or by source statements. Some C compilers allow more than one statement per line, separated by semicolons. You can step through such a source line by statements.

Source Step Granularity sub-menu specifying source line stepping



Step Count opens a dialog box to set the steps (1 to 100) executed per Step command.

Step Count dialog box specifying one step per Step command



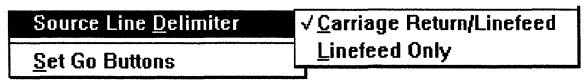
Browser History Depth opens a dialog box to set the maximum number of browsed modules (0 to 50) that can be recalled. The emulator maintains a list of the modules you browse and the order in which you browse them. Use the Previous Browsed Module and Next Browsed Module items in this menu to cycle through the modules.

Previous Browsed Module displays the next earlier module in your browse history.

Next Browsed Module displays the next later module in your browse history.

Source Line Delimiter opens a sub-menu to set the ASCII string used by the compiler to delimit a source line.

Source Line Delimiter sub-menu set for DOS source files

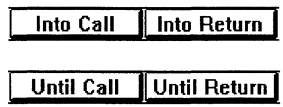


Carriage Return/ Linefeed	(the default) recognizes a carriage return followed by a linefeed as the string indicating the end of a line. This is the DOS standard line delimiter. Displaying a UNIX file with Source Line Delimiter as Carriage Return/Linefeed shows the entire source file as a single line.
Linefeed Only	recognizes a linefeed as the end-of-line indicator. This is the UNIX standard line delimiter. Displaying a DOS source file with Source Line Delimiter set to Linefeed Only shows a black dot at the end of each line.

Set Go Buttons opens a sub-menu to toggle the operation of the Call and Return buttons between Go Until and Go Into.

Set Go Buttons sub-menu specifying Into Call/Return buttons

Into Call/Return and Until Call/Return buttons

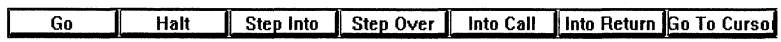


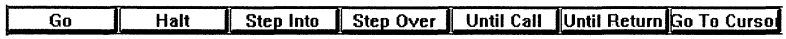
Source Window Buttons

These buttons provide quick access to commonly used Run menu items, described earlier in this chapter.

The Source window button bar has two possible configurations. To toggle between them, choose the Options menu Set Go Buttons item and choose Until Call/Return or Into Call/Return.

Source window button bar configurations





Button	Use To:
Go	Start emulation from the program counter, the same as the Run menu Go.
Halt	Stop emulation, the same as the Run menu Halt.
Step Into	Step into a function call at the program counter, the same as the Run menu Step Into.stepping:Source window
Step Over	Step over a function at the program counter, the same as the Run menu Step Over.
Until Call	Go from the program counter and break before the next

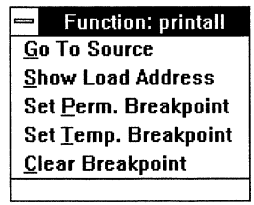
function call, the same as the Run menu Go Until Call.

Into Call	Go from the program counter and break after the next function call, before executing the function, the same as the Run menu Go Into Call.
Until Return	Go from the program counter and break before the next return instruction, the same as the Run menu Go Until Return.
Into Return	Go from the program counter and break after the next return instruction, the same as the Run menu Go Into Return.

Function Popup Menu

To pop-up the Function menu, select a function name in a source line. The selected function name is highlighted.

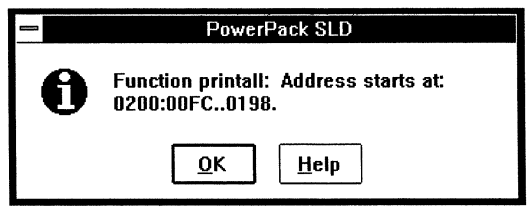
Function menu popped-up by double-clicking on a printall string in the source display



Go To Source puts the Source cursor at the beginning of the function source code. If no source is available, the Source window can display the function in disassembly. To enable the disassembly display, open the View menu and choose Mixed Source and Asm.

Show Load Address opens an information box listing the memory address range occupied by the function.

Load Address dialog box showing the printall load address



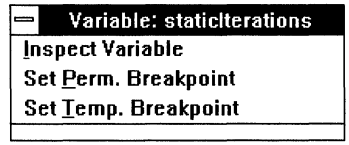
Set Perm. Breakpoint sets a permanent breakpoint at the highlight.

Set Temp. Breakpoint sets a temporary breakpoint at the highlight.

Clear Breakpoint clears the breakpoint at the highlight.

Variable Popup Menu

To pop-up the Variable menu, select (double-click on) a variable name in a source line. The selected variable name is highlighted.



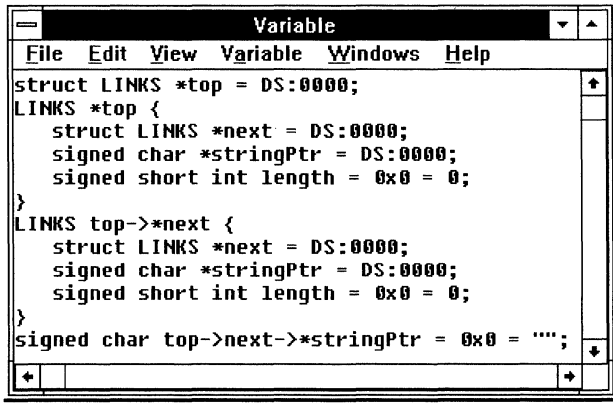
Variable menu, popped-up by double-clicking on a staticIterations string in the source display

Inspect Variable adds the variable to the Variable window, described in the Variable Window Reference chapter. If the Variable window is not already open, Inspect Variable opens it.

Set Perm. Breakpoint sets a permanent breakpoint on the highlight.

Set Temp. Breakpoint sets a temporary breakpoint on the highlight.

Variable Window Reference



Variable Window Contents

The Variable window displays the types, symbolic names, and values of global and local variables. Variable symbolic information appears in the following colors:

Red — indicates an editable value. Integer variables can be edited in hexadecimal or decimal, floating point variables in floating point format, and characters in ASCII or the hexadecimal equivalent. To edit a value, either double-click on the value; or single-click on the value, open the Edit menu, and choose Edit. Press <Enter> to end or <Esc> to cancel editing.

Blue — indicates a pointer variable you can dereference by double clicking. To dereference a pointer, either double click on the pointer name or open the View menu and choose Show. A new entry is added to the Variable window, showing the variable that was pointed to.

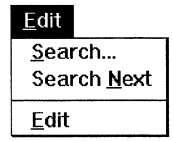
Magenta — indicates a variable. For enum type variables, the enumerated name follows the hexadecimal value.

Variable Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

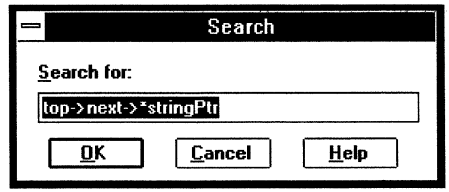
Menu	Use To:
File	Close the Variable window.
Edit	Find and edit a listed variable.
View	Reorganize or refresh the display.
Variable	Add or remove variables from the display.
Windows	Open another SLD window.
Help	Open a window for help with the SLD software.

Edit Menu



Search... opens a dialog box to find a variable in the display. The case-sensitive search stops at the first occurrence or at the end of the display.

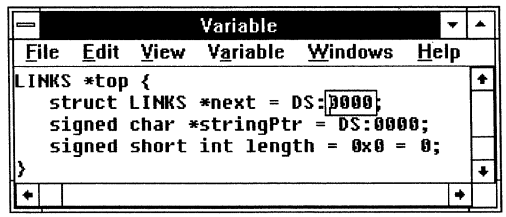
Search dialog box, finding the string top->next->*stringPtr in the Variable window



Search Next finds the next occurrence of the last variable searched for.

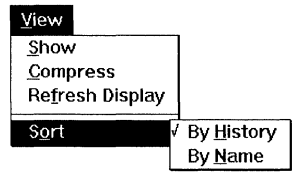
Edit puts an edit field on an editable (red) value. Type a new value in the field. Floating-point values use floating-point format. Characters use hexadecimal or ASCII. Integers use decimal or hexadecimal.

Variable window showing Edit field on the value of the *next symbol



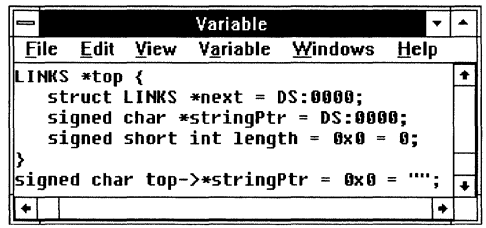
View Menu

View menu showing Sort sub-menu to display variables in the order they are added to the Variable window



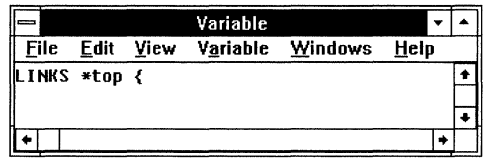
Show adds a line to the Variable window dereferencing the selected variable. This item is available when you have put the Variable cursor on a dereferenceable (blue) symbol, such as a pointer.

Variable window dereferencing the *stringPtr pointer



Compress collapses multi-line variables to the first line.

Variable window in compressed mode



Refresh Display updates the displayed symbols and values.

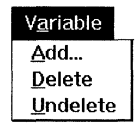
Sort opens a sub-menu to arrange the variables:

By History in the order they were added to the display.

By Variable Name alphabetically.

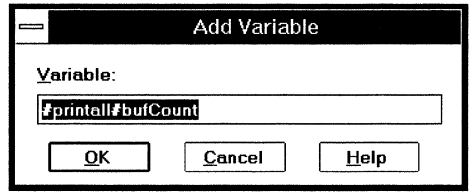
Variable Menu

Variable menu



Add... opens a dialog box to add a variable name to the display.

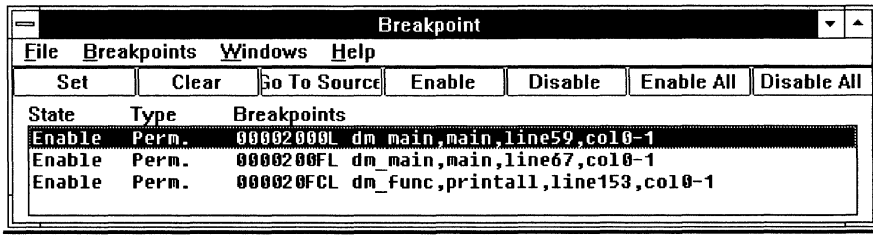
Add dialog box to display the bufCount variable from the printall function



Delete removes the selected variable from the display.

Undelete restores to the display the last variable removed.

Breakpoint Window Reference



Breakpoint Window Contents

The Breakpoint window displays the following information about each breakpoint:

State	Whether the breakpoint will cause a break (Enable) or not (Disable) when emulation executes through the code where the breakpoint is located.
Type	Whether the breakpoint will remain defined (Perm.) or be removed (Temp.) after causing a break.
Breakpoints	The load address, module name, function name, source line number, and source column number where the breakpoint is located. The column number can be affected by the number of spaces the compiler and emulator use for tab characters (the Tab Width).

Breakpoint Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	**Use To:**
File	Exit the Breakpoint window.
Breakpoints	Define, remove, enable, and disable breakpoints.
Windows	Open another SLD window.
Help	Open a window for help with the SLD software.

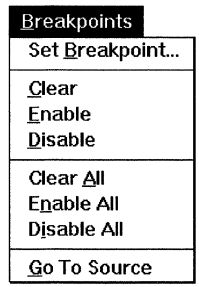
File Menu

Exit closes the Breakpoint window.

Breakpoints Menu

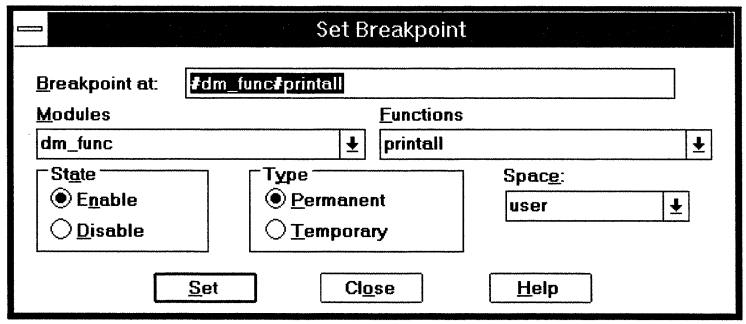
The items available in the Breakpoints menu depend on whether breakpoints are defined and selected. Set Breakpoint... and Go To Source are always available; Clear, Enable, and Disable are available when you have selected a breakpoint from those listed in the Breakpoint window; Clear All, Enable All, and Disable All are available when one or more breakpoints are listed. To select a breakpoint, click on it or use the <Up Arrow> and <Down Arrow> keys to move the highlight.

Breakpoints menu with all items enabled, indicating at least one breakpoint is defined



Set Breakpoint opens a dialog box to define a new breakpoint.

Set Breakpoint dialog box defining a permanent, initially enabled breakpoint at the first instruction of the printall function in the dm_func module



Fill-in the dialog box as follows:

Breakpoint at:	a numeric or symbolic address. You can choose a module and function from the drop-down list boxes.
State	toggled to Enable or Disable. The emulator ignores a disabled breakpoint.
Type	permanent or temporary. A temporary breakpoint is removed after it causes the break.
Space:	User or SMM, depending on the processor.

Choose the Set button to define the breakpoint or the Close button to close the dialog box without defining a new breakpoint.

Clear removes the selected breakpoint.

Disable marks the selected breakpoint to be ignored when emulation executes through the code where the breakpoint is located.

Enable marks the selected breakpoint to cause a break when emulation executes through the code where the breakpoint is located.

Disable All disables, without removing, all breakpoints.

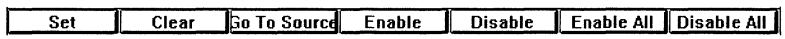
Enable All enables all breakpoints.

Clear All removes all breakpoints. No breakpoints remain defined.

Go to Source opens the Source window, described in the "Source Window Reference" chapter, and positions the source cursor at the specified breakpoint.

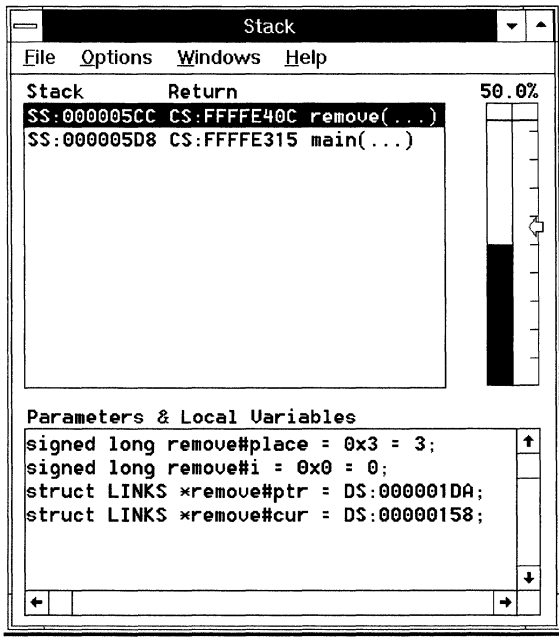
Breakpoint Window Buttons

These buttons provide quick access to commonly used Breakpoints menu items, described earlier in this chapter.



Button	Use To:
Set	Open a dialog box to set a breakpoint, the same as the Breakpoints menu Set Breakpoint... item
Clear	Remove a selected breakpoint, the same as the Breakpoints menu Clear item.
Go To Source	Open the Source window to show the specified breakpoint in source or disassembly, the same as the Breakpoints menu Go To Source item.
Enable	Define that the specified breakpoint will cause a break next time it is encountered in emulation, the same as the Breakpoints menu Enable item.
Disable	Define that the specified breakpoint will cause no break next time it is encountered in emulation, the same as the Breakpoints menu Disable item.
Enable All	Enable all breakpoints, the same as the Breakpoints menu Enable All item.
Disable All	Disable all breakpoints, the same as the Breakpoints menu Disable All item.

Stack Window Reference



Stack Window Contents

The Stack window has three panes:

Frame List	lists the stack address, the return address, and the name of each function on the current call stack. Each line is a stack frame.
Parameters and Local Variables	lists the type, name, and value of each parameter and local variable in the selected stack frame. The format and colors are the same as in the Variable window.
Stack Meter	graphically shows the stack usage statistics, including the percent of the stack area currently in use, an alarm marker at a specified usage level, and a mark at the highest percent usage for the current emulation.

Stack Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	Use To:
File	Close the Stack window; refresh the stack display.
Options	Configure the stack area; toggle the Frame List address display; manage stack usage statistics; inspect the source.
Windows	Open another SLD window.
Help	Open a window for help on the SLD software.

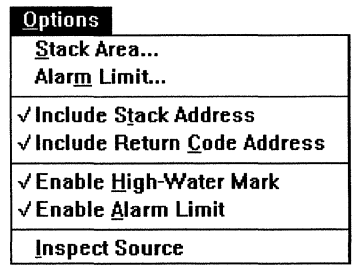
File Menu

Refresh Display reads memory and updates the displayed information.

Exit closes the Stack window.

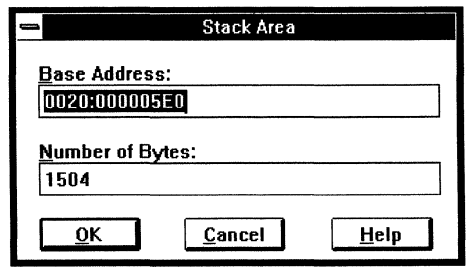
Options Menu

Options menu with all stack statistical options enabled



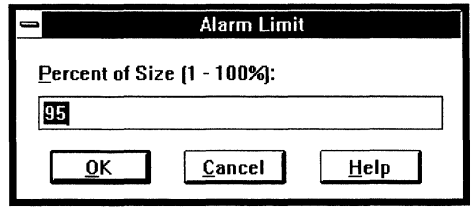
Stack Area... opens a dialog box to set the stack base address and size.

Stack Area dialog box setting the base address and size of the area to be monitored for stack activity



Alarm Limit... opens a dialog box to define the alarm limit as a percentage (1 to 100) of the Stack Meter.

Alarm Limit dialog box setting the alarm at 95% of the monitored stack area



Include Stack Address, when checked, displays stack addresses in the Frame List, in a column labeled Stack. The stack address is the address of the frame in the stack area

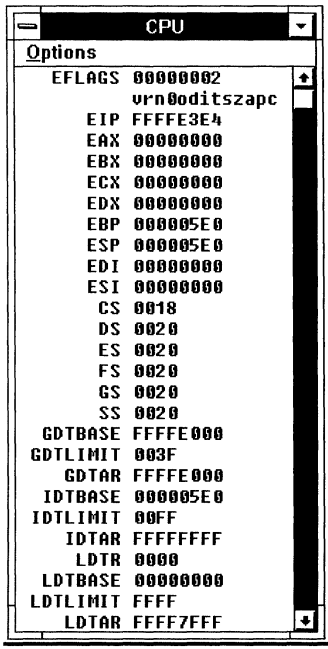
Include Return Code Address, when checked, displays code addresses in the Frame List, in a column labeled Return. The code address is the return address to the calling function.

Enable High Water Mark, when checked, displays the high-water mark on the Stack Meter. The high-water mark indicates the highest percentage that has been used of the stack area.

Enable Alarm Limit displays a warning message each time emulation stops while the alarm limit is exceeded.

Inspect Source opens the Source window, described in the "Source Window Reference" chapter, and positions the Source cursor to show the selected function's source. To select a function, in the Frame List click on the frame or use the <Up Arrow> and <Down Arrow> keys to move the highlight.

CPU Window Reference

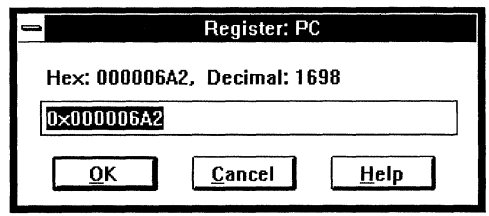


CPU Window Contents

The CPU window lists the processor registers by mnemonic. Different registers appear for different processors, as listed in the *Hardware Reference*. The register values are updated and the changed values highlighted each time emulation halts.

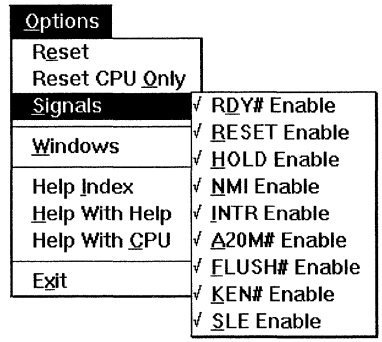
To edit the register values, double-click on a register value; or, move the cursor with <Up Arrow> and <Down Arrow> then press <Enter>.

Register Edit dialog box for changing the EIP register



Options Menu

Options menu showing the EA-486 signals controlled by the emulator



Reset resets and reinitializes the target processor:

- The processor RESET pin is asserted.
- The program counter is read from memory; the Source window is scrolled to the beginning of code.
- The stack pointer is read from memory, resetting the stack; the Stack window display becomes invalid.
- All SLD windows are updated.

Reset CPU Only resets only the processor and does not update the windows. Use Reset CPU Only if Reset fails to reset the processor.

Signals opens a sub-menu to specify whether certain signals are controlled by the target (unchecked) or by the emulator (checked). Different signals can be enabled for different processors. For a list of the signals configurable in your emulator, see the *Hardware Reference*.

Windows opens a sub-menu to open another SLD window. This item is equivalent to the Windows menu in other SLD windows.

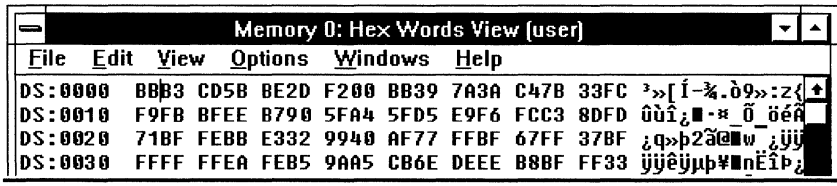
Help Index opens a window with the table of contents for SLD help.

Help With Help opens a window on using a Windows help facility.

Help With CPU opens a window with SLD CPU window help.

Exit closes the CPU window.

Memory Window Reference



Memory Window Menus

The window title identifies the Memory window by number, describes the display format, and identifies the address space. You can have up to 20 Memory windows open simultaneously for a variety of views.

The leftmost column is the address. Address formats differ for different processors. To view another area of memory, double-click in the address column of the Memory window. Enter a numeric or symbolic address in the Go To Address dialog box. Any symbol you enter must have a fixed address; you cannot Go To local variables or stack-resident parameters.

The memory contents can be in disassembly or numeric format. Numeric format shows the hexadecimal or decimal values and, in the rightmost column, the equivalent ASCII values. You can edit memory contents directly in the numeric and ASCII formats by positioning the cursor (a vertical bar) with the mouse, then overtyping the memory display. Disassembly can include symbols; in the Toolbar Configure menu, toggle Symbolic Disassembly.

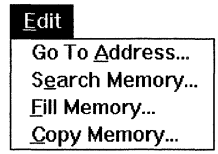
Memory Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	**Use To:**
File	Exit the Memory window.
Edit	Edit memory; navigate the memory display.
View	Choose numeric or disassembly display formats.
Options	Manage memory access options.

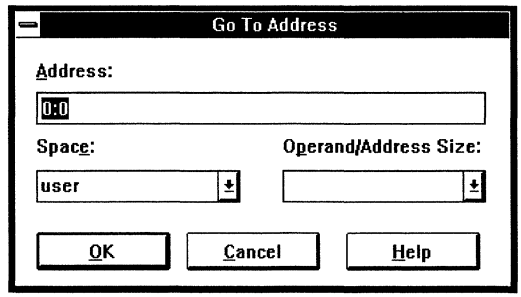
Windows	Open another SLD window.
Help	Open a window for help with the SLD software.

Edit Menu



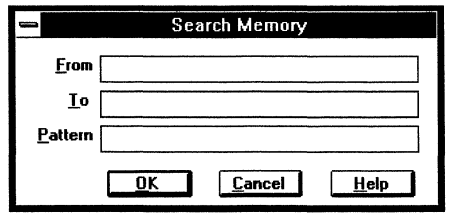
Go To Address... opens a dialog box to change the Memory window display to a specified numeric or symbolic address. For some processors, you can specify User or SMM space.

Go To Address dialog box for finding address 0:0 in user space



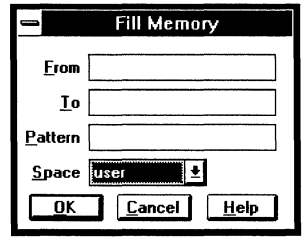
Search Memory... opens a dialog to search a specified address range for a specified pattern. The search stops at the first occurrence of the pattern in the range. If the pattern is not found, the Memory cursor does not move.

Search Memory dialog box for finding a pattern in an address range



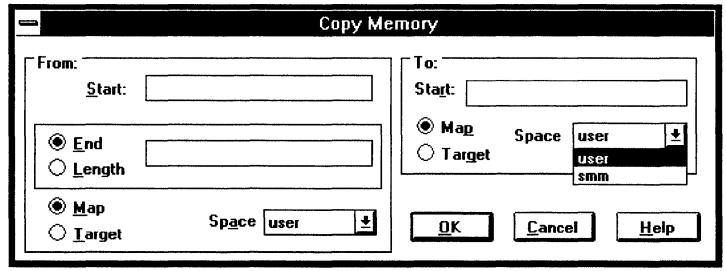
Fill Memory... opens a dialog box to fill an address range with a specified pattern.

Fill Memory dialog box for writing a repeating pattern to an address range



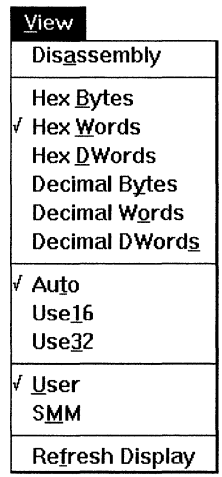
Copy Memory... opens a dialog box to copy one address range to another or to copy target memory to overlay memory.

Copy Memory dialog box for copying memory contents from one address range to another



View Menu

View menu, displaying memory contents as disassembly



Disassembly displays memory disassembled. In Disassembly view, you can double-click on a disassembled line to open the Single Line Assembler dialog box (described later in this chapter).

Hex Bytes displays memory as hexadecimal 8-bit integers with values from 0 to FF.

Hex Words displays memory as hexadecimal 16-bit integers with values from 0 to FFFF.

Hex Dwords displays memory as hexadecimal 32-bit integers with values from 0 to FFFFFFFF.

Decimal Bytes displays memory as decimal 8-bit integers with values from 0 to 255.

Decimal Words displays memory as decimal 16-bit integers with values from 0 to 65,535.

Decimal DWords displays memory as decimal 32-bit integers with values from 0 to 4,294,967,295.

Auto uses the pmode to determine whether operands and addresses are interpreted as 16-bit or 32-bit values. For a description of pmodes, see the *Hardware Reference*.

Use16 interprets operands and addresses as 16-bit values.

Use32 interprets operands and addresses as 32-bit values.

User displays processor user memory.

SMM displays processor system management mode memory (available in some processors).

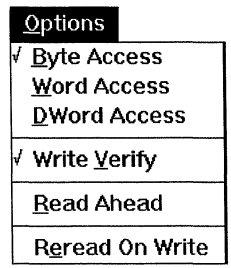
Refresh Display re-reads memory and refreshes the screen. This happens automatically when emulation halts.

To update or scroll the Memory window during emulation, enable Run Access before starting emulation. Check the Toolbar Configure menu Enable Run Access item; or enter a RunAccess Shell command.

Any run-time memory access, such as that used to update the Memory window, takes a small amount of time from the processor and thus can degrade your program performance.

Options Menu

Options menu, specifying 8-bit memory access and verification of memory writes



Byte Access specifies 8-bit cycles for memory access.

Word Access specifies 16-bit cycles for memory access. For writing a byte, the word containing the byte is read, the appropriate byte replaced, and the word re-written. Words at even addresses are read and written as words. Words at odd addresses are read and written as two words. For example, for writing a word of data at an odd address:

1. The word containing the first byte (odd address minus 1) is read.
2. The lower byte of the data is put into the upper byte of the word.
3. The word is re-written at odd address minus 1.
4. The word containing the second byte (odd address plus 1) is read.
5. The upper byte of the data is put into the lower byte of the word.
6. The word is re-written at odd address plus 1.

DWord Access specifies two 16-bit cycles for memory access. Long-word memory writes act as follows:

1. Long-word writes on long-word boundaries use long accesses.
2. Word writes and byte writes read long words, replace the byte or word, and write back as long words.

Set the memory access size to long (dword) for faster loading.

Write Verify, when checked, compares any value written with write, fill, or copy with the expected value and reports discrepancies.

Toggling Write Verify does not affect load verification. Use the verify Shell command to toggle load verification. With Verify=On, a byte read back that does not match the byte written returns an error.

Read Ahead, when checked, reads ahead and caches more data than is displayed in the Memory window screen, for faster scrolling.

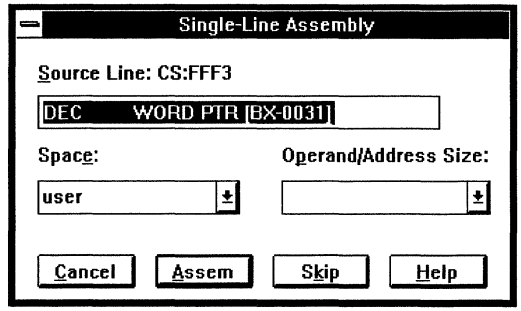
With read-ahead enabled, scrolling through peripheral registers or near invalid memory regions can cause Unterminated Memory Access errors.

Reread On Write, when checked, refreshes the memory display when you edit the numeric or ASCII fields in the display. Toggling Reread On Write does not affect Memory window refreshing for memory changes done outside of the memory display. For example, load, fill, write, and copy operations always refresh the memory display.

Single-Line Assembler Dialog Box

You can patch code into memory an assembly-line at a time with the single-line assembler. With the Memory window in Disassembly view, double-click on the line you want to replace.

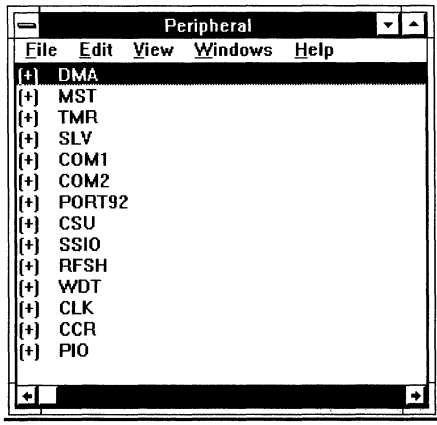
Single-Line Assembly dialog box, assembling a DEC instruction at location CS:FFF3



Type a line of assembly language in the text box.

Source Line:	shows the address where the line will be assembled.
Space:	for some processors, can be User or SMM.
Operand/ Address Size:	is unavailable.
Cancel	closes the single-line assembler dialog box without assembling. Once you have assembled a line, this button changes to Done. Choosing Done closes the dialog box; your assembled changes remain in memory.
Assem	assembles the line into memory; advances the address.
Skip	advances the address without assembling the line.
Help	opens a window for help on the single-line assembler.

Peripheral Window Reference



Peripheral Window Contents

Different peripherals are supported for different processors.

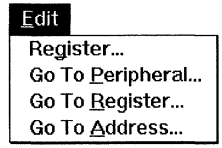
The Peripheral window shows the peripheral register information heirarchically. Click on the (+) or (-) at the left of a line to expand or collapse the hierarchy. At the top level (the only level visible when the heirarchy is fully collapsed) are the peripherals. Expanding a peripheral shows its registers. Expanding a register shows its bit fields. Full expansion lists the register address, bit field bit position, value, name (mnemonic), and description.

Peripheral Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	**Use To:**
File	Exit from the Peripheral window.
Edit	Edit a register; navigate the Peripheral display.
View	Refresh, expand, or compress the display.
Windows	Open another SLD window.
Help	Open a window for help with the SLD software.

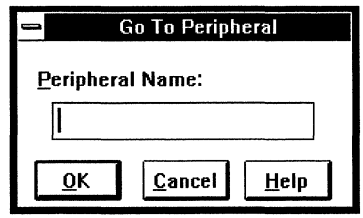
Edit Menu



Register... opens a Register Edit dialog box (described later in this chapter) to edit the selected register. To select a register or bit field, use the mouse or <Up Arrow> and <Down Arrow> keys to move the highlight. Selecting a peripheral selects its the first register.

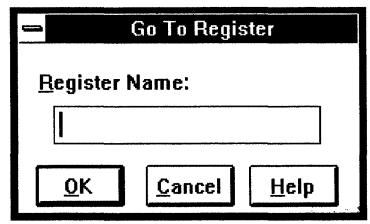
Go To Peripheral... opens a dialog box to scroll to the peripheral specified by name.

Go To Peripheral dialog box for finding a peripheral by name



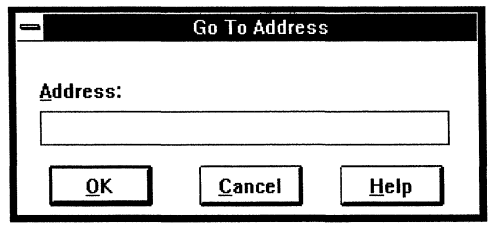
Go To Register... opens a dialog box to scroll to the register specified by name.

Go To Register dialog box for finding a peripheral register by name



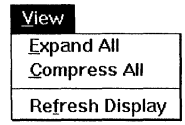
Go To Address... opens a dialog box to scroll to the register specifed by address.

Go To Address dialog box for finding a peripheral register by hexadecimal address



View Menu

View menu



Expand All expands the hierarchy completely, showing all peripheral, register, and bit field mnemonics, with the addresses or bit positions, values, and descriptions of the registers and bit fields.

Compress All collapses the hierarchy completely, showing only the peripheral mnemonics.

Refresh Display re-reads the readable registers and refreshes the screen. This also occurs automatically when emulation halts.

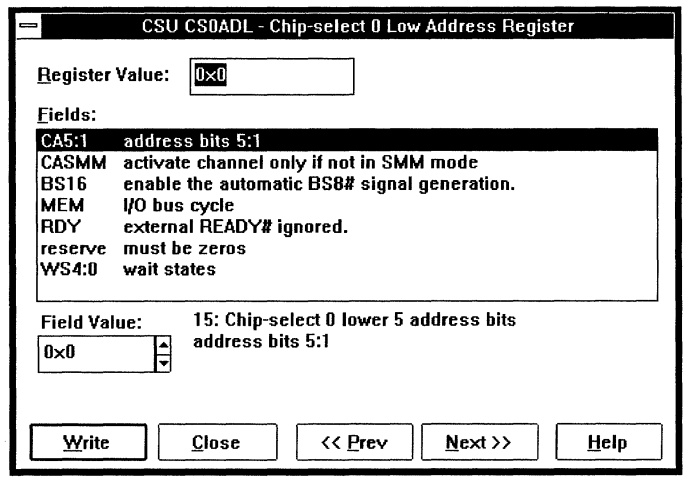
To update or scroll the Peripheral window during emulation, enable Run Access before emulating. Check the Toolbar Configure menu Enable Run Access item; or enter a RunAccess Shell command. Any run-time memory access, such as that used to update the Peripheral window, takes a small amount of time from the processor and thus can degrade your program performance.

Write-only register fields display the most recent value you entered using the Peripheral or Shell window interface. Values written by program execution are not captured by the emulator.

Register Edit Dialog Boxes

Different registers have different field values.

Register Edit dialog box for changing the CS0 low address register value



Register Value shows the register contents in hexadecimal. You can edit this field.

Fields lists each bit field mnemonic in the register and its effect on the processor. To select a bit field, click or use the <Up Arrow> and <Down Arrow> keys to move the highlight.

Field Value is a spin box showing the value of the bit field selected in the Fields box. You can edit this field. To ensure you enter an acceptable value for the bit field, click on the spin arrows or use the <Up Arrow> and <Down Arrow> keys to change the value. Editing the Field Value changes the Register Value.

The selected bit field position and a description of the bit field according to its current value are listed under the Fields box, to the right of the Field Value spin box. This description changes when you change the bit field value.

Write writes the value shown in Register Value:.

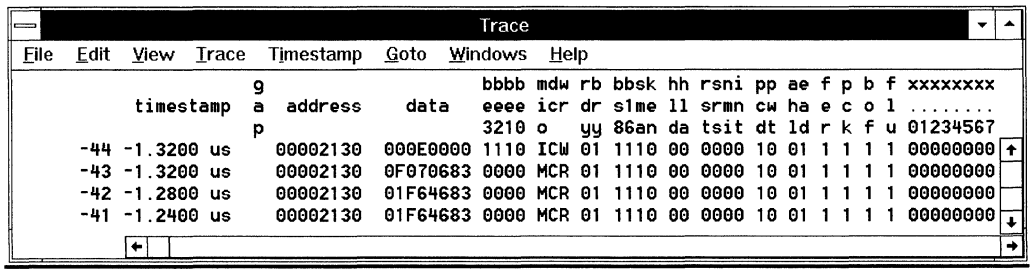
Close closes the Register Edit dialog box.

<<Prev displays the Register Edit dialog box for the previous register in the Peripheral window list.

Next>> displays the Register Edit dialog box for the next register in the Peripheral window list.

Help opens a help window on the Register Edit dialog box.

Trace Window Reference



Trace Window Contents

Trace features differ between the PP, EA, and SW emulators. Different signals are available for different processors. Grayed-out menu names and items indicate unavailable features.

The Trace window has three views:

Bus	displays every cycle of bus activity.
Clock	(PP, EA) displays address, data, and processor status signals aligned on clock cycles.
Instruction	displays disassembled instructions. The first instruction must follow a change in execution flow.

Each trace frame (one line in the Trace window) contains the following information, in columns from left to right:

Frame number	The number of the trace frame relative to the clock cycle on which tracing stopped. The frame number increments by one for each captured frame. For unqualified trace, a frame is captured on each clock or bus cycle. EA qualified trace captures a frame on each cycle meeting the qualification criteria. In instruction and bus views, the frame numbers are discontinuous because multiple clock cycles make up a single bus or instruction frame.
Timestamp	(PP, EA) The time the trace frame occurred, relative to a specified frame or time.
Address	The value on the address bus.

In bus or clock view:

Data	The value on the data bus
Signals	The values of processor-specific signals. The signal mnemonic labels are formatted vertically. For a list of traced signals, see the *Hardware Reference*.

In instruction view, disassembly is shown instead of data and signals. Also, the number of clock cycles between instruction frames describes how many cycles have elapsed between signals appearing on the target processor external pins (for example, the number of cycles between successive prefetches); this number does not, for example, report how many clocks the processor used to execute an instruction.

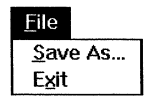
Trace Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	Use To:
File	Save trace to a buffer; close the Trace window.
Edit	Open the Event window; search for an event; clear trace.
View	Configure the trace display; link the Source window display to scroll with the Trace window cursor.
Trace	Control and configure trace capture.
Timestamp	Configure the timestamp and the system clock frequency.
Goto	Navigate through the Trace buffer.
Windows	Open another SLD window.
Help	Open a window for help on the SLD software.

File Menu

File menu



Save As... opens a dialog box to save the trace buffer to a file. Enter the filename. If a file with the specified name already exists, it will be overwritten. The (Trace) Save As dialog box differs between emulators:

File Name:	is the drive, directory, and filename you specified in the first dialog box. You can edit this string.
Save Format	(PP) saves the trace in bus, clock (when available), or instruction format.

Buffer	(PP) with multiple buffers configured, saves a specified range of buffers.
Frame	(PP) saves a specified range of frames.

Exit closes the Trace window.

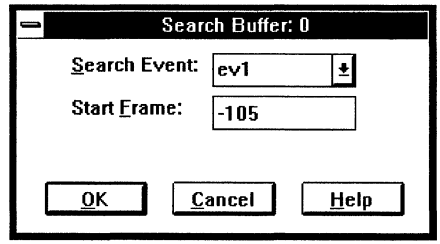
Edit Menu

PP Edit menu, with Events and Search items available

Events... opens the Event window.

Search... opens a dialog box to find an event in the currently displayed trace buffer. For PP emulators, the title bar shows the buffer searched.

PP Search Buffer dialog box to search for event ev1 in trace buffer 0, from frame -105 to the end of the buffer



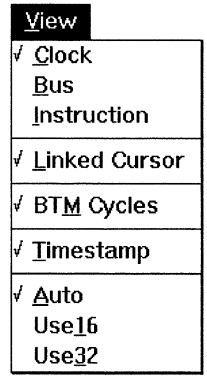
Search Event	select an event from the list of defined events.
Start Frame	select the frame to start searching.

Clear Trace clears all trace buffers and resets the buffer pointer to zero. (The current trace buffer is automatically cleared and reset when you start emulating or tracing.)

View Menu

Different items are available for different emulators.

PP view menu displaying trace as clock cycles, displaying timestamps, and collecting BTM cycles to allow disassembly (Instruction instead of Clock view)



Clock (PP, EA) displays trace as clock cycles. Capture trace in clock mode for clock cycle display.

Bus displays trace as bus cycles. Trace captured in bus mode can be displayed only as bus cycles.

Instruction displays trace as disassembly . Capture trace in clock mode and capture branch trace messages for instruction mode display. Frames prior to any such messages cannot be disassembled. To capture branch trace messages, in the PP enable the View menu BTM cycles item or in the EA enable the Trace menu Trace Capture dialog box Instruction Mode Assist.

Linked Cursor synchronizes the Source and Trace window cursors, so scrolling the Trace window displays the corresponding code in the Source window. This feature is available only in instruction view.

BTM Cycles (PP) generates Intel-x86 BTM cycles and collects them in trace. A BTM cycle is a special bus cycle executed by the Intel bondout processor when execution is discontinuous (e.g., at a jump, call, interrupt, or return). BTM generation degrades real-time execution slightly. For trace to be displayed as instructions, BTM cycles must be collected. Toggling BTM Cycles clears the trace buffer.

Timestamp displays the timestamps.

Auto uses the pmode to determine whether operands and addresses are interpreted as 16-bit or 32-bit values.

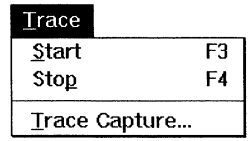
Use16 interprets operands and addresses as 16-bit values.

Use32 interprets operands and addresses as 32-bit values.

Trace Menu

The Trace menu differs between emulators.

EA Trace menu, with Trace Capture item instead of PP Trace Control

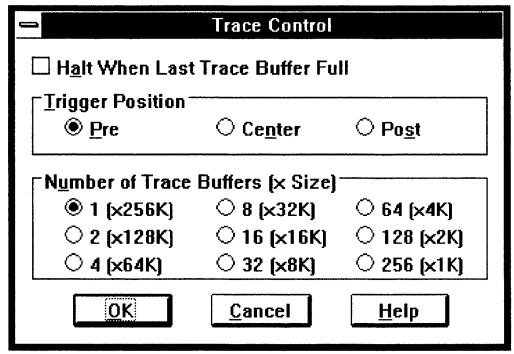


Start (or pressing the F3 key) starts trace collection. This occurs automatically when emulation begins.

Stop (or pressing the F4 key) stops trace collection.

Trace Control... (PP) opens a dialog box to configure how trace information is collected.

PP Trace Control dialog box with one buffer configured and the trigger positioned near the end of the buffer

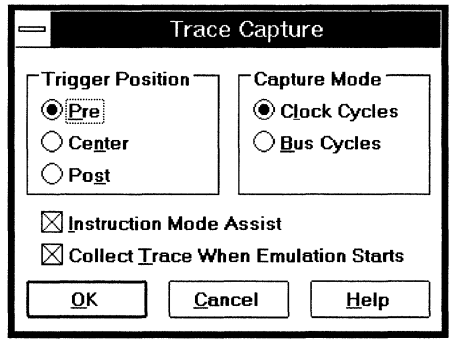


Halt When Last Trace Buffer Full	stops emulation after the last trace buffer has been filled. This overwrites the first trace buffer.
Trigger Position	positions the trigger frame in the trace buffer:
	Pre — collects cycles before the trigger. The event appears near the end of the buffer.
	Center — collects cycles before and after the trigger. The event appears in the middle of the buffer.
	Post — collects cycles after the trigger. The event appears near the beginning of the buffer.
Number of Trace Buffers (x Size)	with 256 bytes of trace memory installed, configures 256 trace buffers each of which is 1K byte (512 frames) long, or a single trace buffer 256K bytes long, or any of various combinations in between. With 128K bytes of trace memory, you have the same choices for number of buffers but each buffer is half the size.

Trace Capture... (EA) opens a dialog box to configure how trace information is collected.

Trace Capture dialog box to:

- position the trigger as the last frame
- collect trace as clock cycles
- include branch trace messages
- start tracing when emulation starts



Trigger Position — positions the trigger frame (the frame matching a Trigger window condition with a Trig action) in the trace buffer:

- Pre — saves any frames before the trigger to the limit of the trace buffer and stops trace after the trigger.
- Center — stops trace 125000 clocks after the trigger.
- Post — stops trace 250000 clocks after the trigger.

For Center and Post collection, the number of frames collected after the trigger depends on whether trace is initially on, qualified, or turned on or off during the time limit. Frames before the trigger are lost only as needed to make room for frames after the trigger.

Capture Mode — collects trace as clock or bus cycles. Trace collected as bus cycles can be viewed only as bus cycles

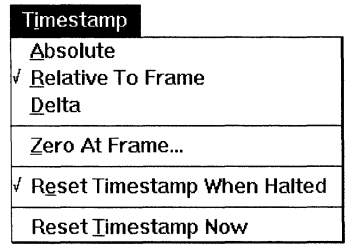
Instruction Mode Assist — collects branch trace messages generated when execution flow is discontinuous. Such messages provide address synchronization necessary for disassembly.

Collect Trace When Emulation Starts — starts trace collection when emulation starts rather than waiting for a manual start or trigger action.

Timestamp Menu

Timestamp is available for PP and EA emulators.

EA Timestamp menu calculating timestamps relative to a base frame specified in Zero At Frame



Relative To Frame shows timestamps as elapsed time from a base frame specified in Zero At Frame.

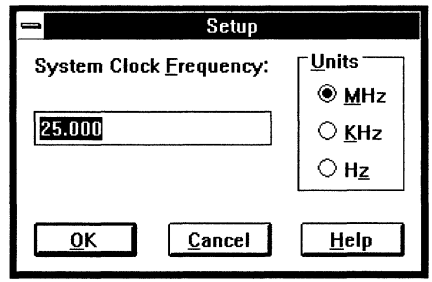
Delta shows timestamps as incremental time between frames.

Absolute (EA) shows timestamps as elapsed time from the last timestamp reset.

Zero At Frame... sets the base frame for calculating the Relative To Frame timestamp. In the trace display, the zero frame is marked with dashes (--).

Setup... (PP) opens a Setup dialog box to set the system clock frequency, used in calculating the PP timestamp. Enter a floating-point value from 0.01 Hz to 40 MHz.

PP Setup dialog box specifying the system clock frequency as 25.000 MHz



The EA timestamp increments at a constant rate of 33 MHz, independent of the system clock.

Reset Timestamp Now (EA) resets the timestamp to 0.

Reset Timestamp When Halted (EA) resets the timestamp every time emulation halts.

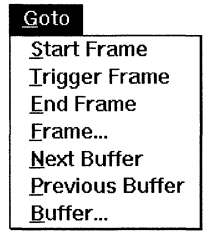
Goto Menu

The Trigger Frame is available for EA and PP emulators.

Multiple buffer navigation (Next Buffer, Previous Buffer, and Buffer...) is available for PP emulators when multiple buffers are configured. Frame navigation (Start Frame, Trigger Frame, and End Frame) apply to the current buffer.

EA emulator qualified trace captures separate frames or blocks of frames in a single buffer rather than separate blocks of frames in separate buffers. Use frame navigation to find specific events.

PP Goto menu with multiple buffers configured



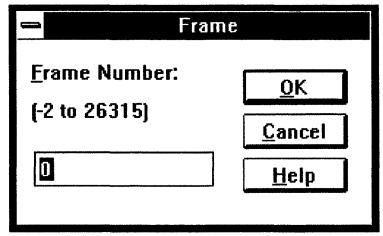
Start Frame scrolls to the first frame.

Trigger Frame (EA, PP) scrolls to the trigger frame.

End Frame scrolls to the last frame.

Frame... opens a dialog box to scroll to a specified frame in the displayed trace buffer.

Frame dialog box for finding a frame number

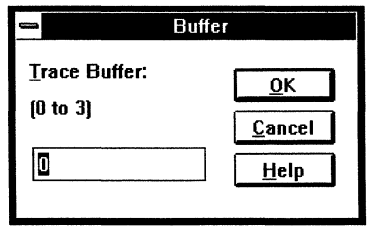


Previous Buffer (PP) displays the next lower numbered buffer.

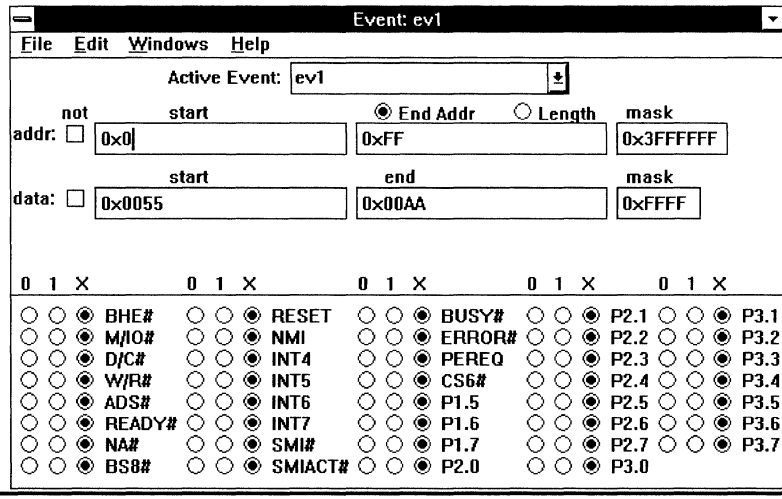
Next Buffer (PP) displays the next higher numbered buffer.

Buffer... (PP) opens a dialog box to display the specified buffer.

PP Buffer dialog box, with four buffers configured, for displaying buffer 0



Event Window Reference



Event Window Contents

Events are available for EA and PP emulators. Different signals are available for different processors, as listed in the *Hardware Reference*.

The Event window defines an event to be used as a condition for triggering or as a pattern for searching in trace. The fields are:

Active Event — names the event described in the Event window. This name identifies the event in the Trigger and Trace windows.

addr — describes a single address or range of numeric or symbolic addresses. Select End Addr to specify the last location in a range or Length to specify the number of bytes in the range.

data — describes a data value or range of data values.

mask — is a hexadecimal value to be bitwise-ANDed with the described addresses or data. Use all F's to include all contiguous values in the described range. Vary the mask to describe a discontinuous pattern of values.

not — when checked, defines the event as any memory access that does not match the described range or pattern.

0 1 X	specifies each signal value as low (0), high (1), or don't-care (X). Active-low signals are shown with a hash mark (#). The signals available depend on the target processor.

Event Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	Use To:
File	Save and restore events in files; close the Event window.
Edit	Add, delete, and redefine events.
Windows	Open another SLD window.
Help	Open a window for help with the SLD software.

File Menu

File menu

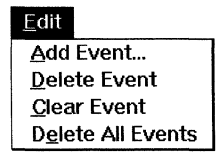
Save Events As... opens a dialog box to save the events to a file.

Restore Events... opens a dialog box to add events from a previously saved file. Currently defined events are not deleted; events with duplicate names are overwritten from the file.

Exit closes the Event window.

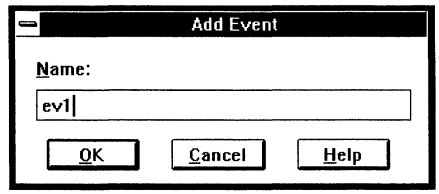
Edit Menu

Edit menu



Add Event... opens a dialog box to create a new event. Enter the name of a new event in the box and choose OK. The new event then appears as the Active Event, with all fields cleared, in the Event window.

Add Event dialog box
for naming a new event

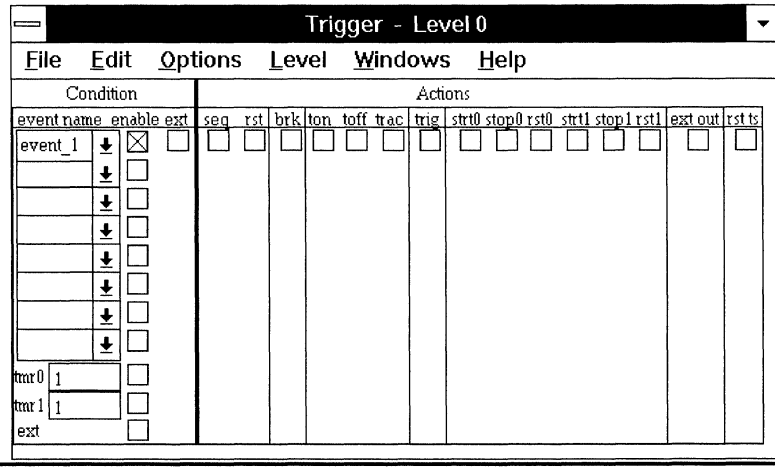


Delete Event deletes the currently displayed event.

Clear clears the event definition fields without deleting the event name

Delete All Events deletes all currently defined events.

Trigger Window Reference



Trigger Window Contents

Triggers are available for PP and EA emulators.

The Trigger window has two panes:

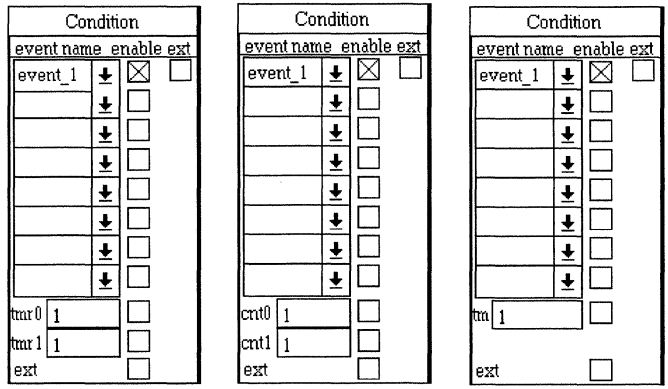
Condition — describes one or more conditions, including events, an external trigger-low signal, and either two counter values or a timer value.

Actions — specifies one or more actions to be taken for each condition met during emulation. When multiple conditions are met simultaneously, all associated actions are taken.

The title bar displays a level number from 0 to 3. The level 0 trigger is enabled when you start emulation. Each trigger can, as one of its actions, disable itself and enable the next level trigger. Thus you can define up to four sequential triggers.

Condition Fields

Trigger condition fields showing paired timers, paired counters, or a single timer



event name — Select an event by the name defined in the Event window. You can use up to 8 events per trigger. If no event is defined when you click on an event name condition, the Event Name dialog box appears for defining a new event.

enable — Activate a condition. You can define several conditions and actions, then vary your triggering scheme by enabling them in different combinations.

ext — (This is the ext that appears when an event, timer, or counter is enabled.) Specify that the condition must occur at the same time as an external trigger signal. The PP external trigger is active-low. Set the EA external trigger input active-low or active-high with the Options menu Trigger In Active items.

cnt0/1 — To configure the PP Trigger window for a pair of 10-bit counters (each with a value range of 1 to 1023), enable the Options menu Counter item. To configure the EA Trigger window for a pair of 16-bit counters (each with a value range of 1 to 65535), enable the Options menu 2 Counters item.

Type a target value in a counter field and enable the counter. Trigger actions can reset (to 1) or increment (by 1) the counter. When the count matches the specified number, the counter condition is met and the associated actions occur.

tmr0/1, tmr — To configure the EA Trigger window for a pair of 10-bit timers, enable the Options menu 2 Timers item. Each timer has a value range of 1 to 1023 clock cycles.

To configure the PP Trigger window for a single 20-bit timer, enable the Options menu Timer item. To configure the EA Trigger window for a single 32-bit timer, enable the Options menu Cascaded Timer item. The timer has a value range of 1 to 1,048,575 (PP) or 4,294,967,295 (EA) clock cycles.

Type a target value in a timer field and enable the timer. Trigger actions can start counting clock cycles from the current number; stop counting without resetting the timer; or reset the timer to 1. You can combine resetting with either starting or stopping the PP timer; for such combinations in the EA, define two identical conditions. When the timer count matches the specified time, the timer condition is met and the associated actions occur.

The timer increments at the clock rate of the emulation processor and wraps to 0 after reaching its maximum value. To calculate how much time is represented by a complete cycle of the timer, use:

PP_wrap_time = (2^{20}) / (clock_period)

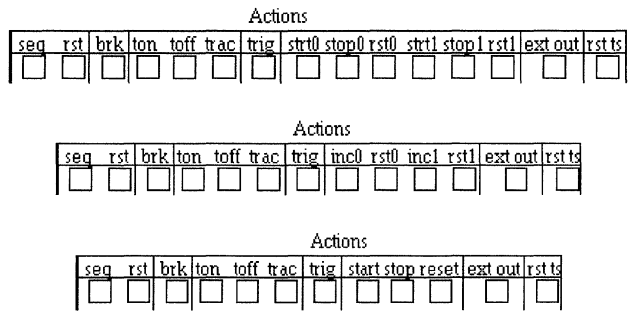
EA_wrap_time = (2^{32}) / (clock_period)

For example, at 25 MHz, the PP timer wraps in about 42 ms; at 16 MHz, in about 65.5 ms. The EA timer, always at 33 MHz, wraps in about 128 seconds.

ext — (This is the ext in the lower left corner of the Trigger window.) Detect an external trigger signal. The PP external trigger is active-low. Set the EA external trigger input active-low or active-high with the Options menu Trigger In Active items.

Action Fields

EA trigger action fields showing controls for paired timers, paired counters, or a single timer



seq — Disable the current trigger and enable the next level trigger.

rst	Disable the current trigger and enable the level 0 trigger.
brk	Halt emulation.
toff	(EA) Suspend trace until another tracing command or action.
toff	(PP) Fill the current buffer according to the Trace Control dialog box settings, then turn trace off until emulation is halted and restarted.
next	(PP) With multiple buffers defined, fill the current trace buffer according to the Trace Control dialog box settings, then start collecting trace in the next buffer.
ton	(EA) Start trace.
trig	(EA) Fill the trace buffer according to the Trace Capture dialog box settings, then turn trace off until emulation is halted and restarted.
trac	(EA) Collect the current bus or clock cycle.
inc0/1	Increment ctr0 or ctr1 by 1.
rst0/1	Reset ctr0 or ctr1 to 0.
start	Start tmr from its current value.
stop	Stop tmr at its current value.
reset	Reset tmr to 0.
strt0/1	(EA) Start tmr0 or tmr1 from its current value.
stop0/1	(EA) Stop tmr0 or tmr1 at its current value.
rst0/1	(EA) Reset tmr0 or tmr1 to 0.
ext out	(EA) Put a low, high, or open-collector value on the external trigger signal. Set the EA external trigger output with the Options menu Trigger Out items.
ext lo/hi	(PP) Put a low or high value on the external trigger signal.
rst ts	(EA) Reset the timestamp.

Trigger Window Menus

Some items are on/off toggles, on when a check mark (✓) appears. Others take immediate action. Items with ellipses pop-up dialog boxes.

Menu	**Use To:**
File	Exit the Trigger window.
Edit	Specify an event using the Event window.

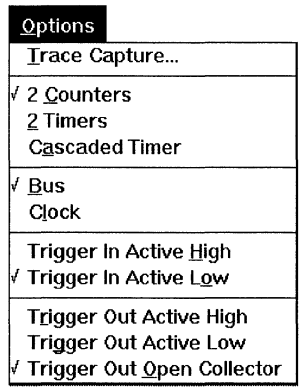
Options	Configure the trace buffers; toggle counter/timer conditions and actions; toggle bus/clock cycle triggering.
Level	View a specified trigger level.
Windows	Open another SLD window.
Help	Open a window for help with the SLD software.

Edit Menu

Events... opens the Event window

Options Menu

EA Options menu configured for conditions with counters and an active-low external trigger input; and for actions with open-collector external trigger output and bus-cycle event recognition



Trace Control... (PP) or **Trace Capture...** (EA) opens the Trace Control or Trace Capture dialog box, described in the "Trace Window Reference" chapter.

Counter (PP) or **2 Counters** (EA) configures two 10-bit (PP) or 16-bit (EA) counters for use in trigger conditions and actions.

Timer (PP) or **Cascaded Timer** (EA) configures a 20-bit (PP) or 32-bit (EA) timer for use in trigger conditions and actions.

2 Timers (EA) configures two 10-bit (PP) or 16-bit (EA) timers for use in trigger conditions and actions.

Bus lets the trigger recognizes conditions on valid bus cycles only. Choose Bus mode except when:

- tracking hardware bus problems possibly caused by processor cycles between valid address, data, or status cycles
- triggering on the initial transition of a hardware signal

Clock uses clock cycles as trigger conditions. Address, data, and status events occur at different clocks. Chose Clock mode for a single event that tests conditions including address, data, and status.

Trigger In Active High (EA) configures the external trigger input active high.

Trigger In Active Low (EA) configures the external trigger input active low.

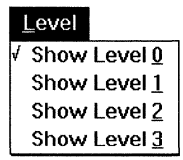
The **Trigger Out** (EA) items configure the external trigger output:

Menu Item	Active	Inactive
Trigger Out Active High	+5V	GND
Trigger Out Active Low	GND	+5V
Trigger Out Open Collector	GND	Resistor pull-up to +5V

Level Menu

Choosing a level displays the conditions and actions for that trigger.

Level menu showing the first level



Level 0 shows the triggers active when emulation starts or after a Rst action.

Level 1 shows the triggers active after a Level 0 trigger's Seq action.

Level 2 shows the triggers active after a Level 1 trigger's Seq action.

Level 3 shows the triggers active after a Level 2 trigger's Seq action.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

MICROTEK INTERNATIONAL, INC
Development Systems Division
3300 N.W. 211th Terrace
Hillsboro, OR 97124-7136
Phone: (503) 645-7333
Fax: (503) 629-8460
Email: info@microtekintl.com
Web: http://www.microtekintl.com

6, Industry East Road 3
Science-Based Industry Park
Hsinchu 30077
Taiwan, ROC
Phone: +886 35 772155
Fax: +886 35 772598
Email: easupport@adara1.adara.com.tw

SLD™ Source Level Debugger for the PowerPack® In-Circuit Emulator
for x86 Target Processors
User's Manual
Part Number 15055-000